Design Document

Dormouse Project

Team #8C: David Gomez-Gomez, Dylan Jean-Baptiste, Adam Rappaport

Team Lead: David Gomez-Gomez

Date: December 6, 2022

Table of Contents

Revision Record

Date	**Author**	**Comments**
Sep 6, 2022	Team	Document Created (system design and hierarchical design)
Sep 20, 2022	David Gomez-Gomez	Software Documentation added
Sep 24, 2022	David Gomez-Gomez	Updated Software Documentation
Sep 24, 2022	Dylan Jean-Baptiste	Updated Software State Machine
Oct 20, 2022	Dylan Jean-Baptiste	Updated Top-Level Design Diagram
Oct 20, 2022	David Gomez-Gomez	Updated Integration Section
Oct 20, 2022	Adam Rappaport	Updated Motion Subsystem Diagram
Oct 23, 2022	David Gomez-Gomez	Updated Software Design Section
Oct 23, 2022	Dylan Jean-Baptiste	Updated Electrical Subsystem Diagram
Oct 23, 2022	Adam Rappaport	Updated Power Subsystem Diagram
Oct 23, 2022	Adam Rappaport	Updated Team Schedule
Nov 10, 2022	Dylan Jean-Baptiste	Updated Electronic Design
Dec 1, 2022	David Gomez-Gomez	Finished Integration Section
Dec 3, 2022	David Gomez-Gomez	Added Conclusion Section
Dec 3, 2022	Adam Rappaport	Updated Team Schedule
Dec 6, 2022	Adam Rappaport	Updated Mechanical Design

Project Description

We have designed a teapot with the character Dormouse from Alice in Wonderland who will pop out of a teapot whenever he receives instruction to do so from the director. The teapot will have the ECE logo and LED lights that synchronize with the instructions received from the director. Below the teapot will be a plate that features 'on' and 'off' buttons as well as a speaker and a switch for testing. Audio, lighting, and movement will be synchronized.

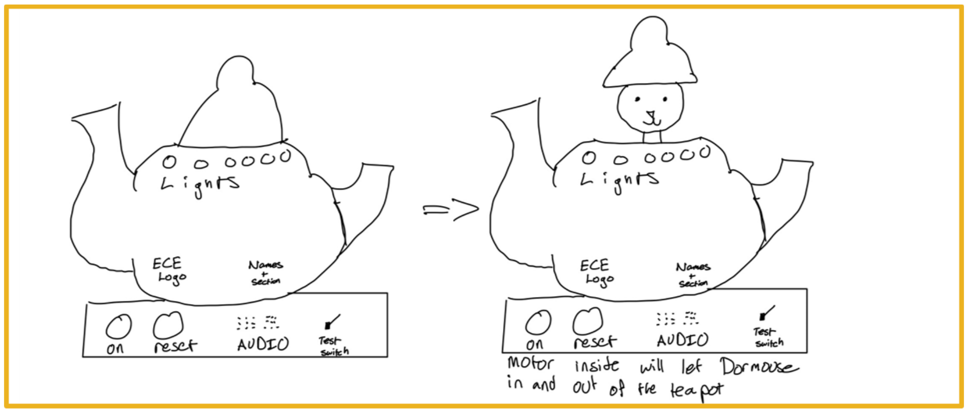


Figure 1: Our initial sketch of Dormouse

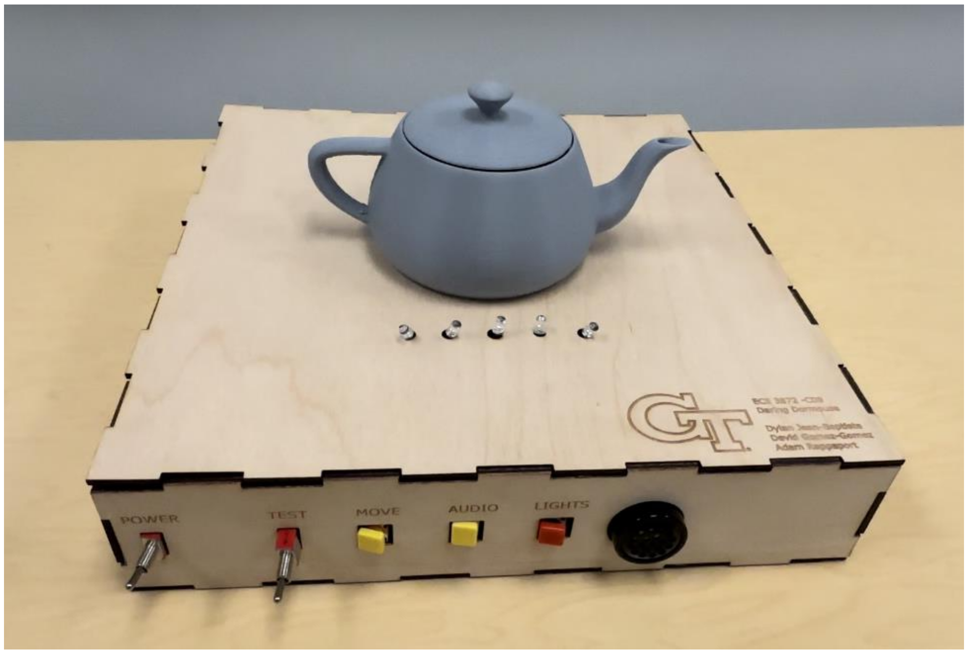


Figure 2: Our finished Dormouse Project

System Design

The system's inputs include 120V AC input power and user input in the form of buttons and a switch. The outputs of this system include a speaker and a motor used to have Dormouse enter and exit the teapot. The system consists of four subsystems including software, power, motion, and electrical. The electrical subsystem includes audio and lighting setup using PWM input and a speaker, and LEDs, respectively. The software subsystem consists of a Raspberry Pi processor. The motion subsystem will use a motor that controls a scissor lift which will raise Dormouse when necessary. The power subsystem includes a 120V AC to 5V DC transformer, a user input switch, and a fuse.

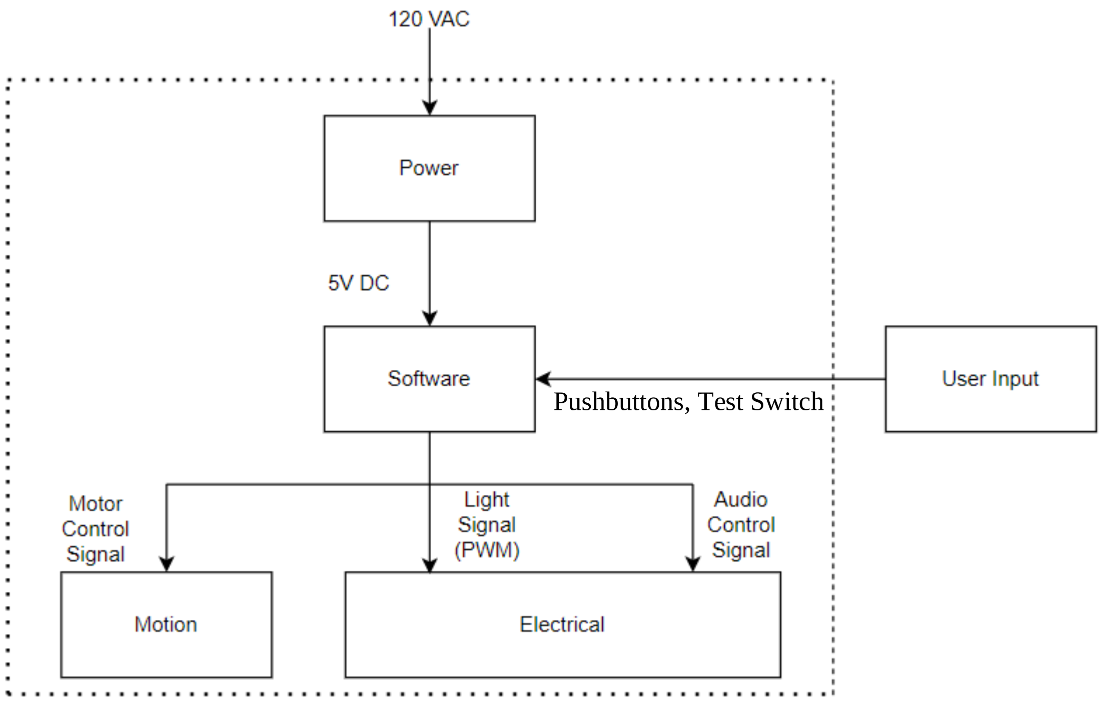


Figure 3: System Diagram

Table 1: Sub-system inputs and outputs

Input/Output	Interface
120 VAC	AC Voltage from standard wall outlet. Will be stepped down by Power Subsystem
5V DC	DC Voltage from power supply. Used to power software subsystem.
Motor Control Signal	GPIO signal that will turn on motor. The signal will also specify which direction motor will spin
Light Signal (PWM)	PWM signal that will power LEDs. This can be a blinking effect or constant light.
Audio Control Signal	GPIO signal that will send specific audio to speaker.

Operating the Robot

The robot should first be plugged into the wall and the test switch should be flipped up. Due to the lack of a director, the robot will not work if the switch is down. Next the power switch can be flipped up. The robot will need about 25 seconds to initialize. During this time the LEDs will be on very dimly. Once the lights are completely off, the test buttons should be fully functional. The robot very rarely encounters a bug where the buttons do not work, even after the conditions above are met. To fix this issue, the robot's power should be switched off and on again. Once 25 seconds pass, the robot should be fully functional.

Sub-System Designs

Our design incorporates four sub-systems: software, motion, power, and electrical. The electrical will focus on the robot's audio and lighting portion, the motion will execute the figure's movement, and the power will convert 120V AC to 5V DC. The software subsystem will integrate all the sub-systems by controlling them via Python code.

Electrical Sub-System

The audio system will take 5 Volts DC and receive an analog signal from the processor in the software system and then amplify it in the class D amplifier and then play it through the speaker. The LED circuit will be powered by 5 Volts DC and will receive a digital signal to turn the circuit on.

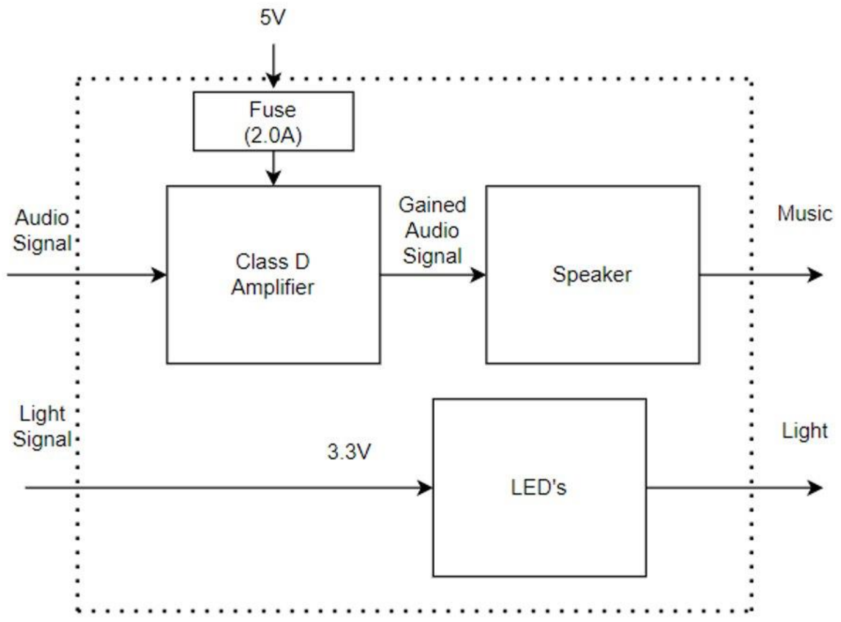


Figure 4: Electrical Subsystem

Motion Sub-System

The motion subsystem will contain a switch activated when it receives a digital signal from the Software Subsystem. When powered, the switch will then send a 5V signal to a DC motor. This motor will power a scissor lift, which will move Dormouse in and out of the teapot.

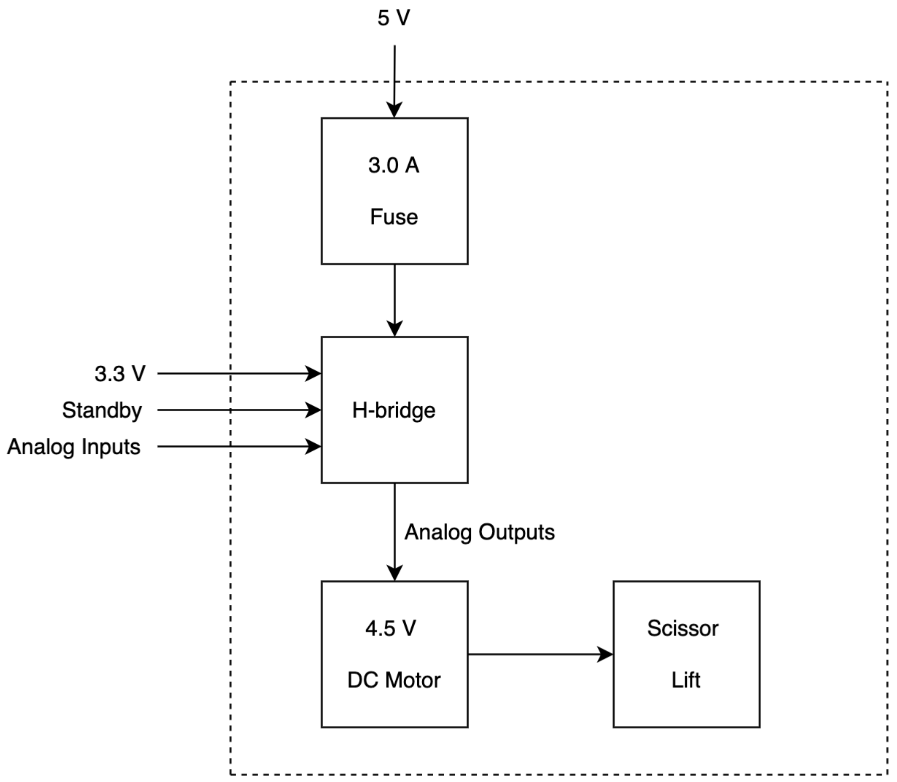


Figure 5: Motion Subsystem

Power Sub-System

The power subsystem will convert 120 Volts AC input and convert it to 5 Volts DC output. It will then send power through a 20-Watt transformer, which then flows through a user input toggle switch and an 8.0-Amp fuse before distributing it elsewhere to the other subsystems.

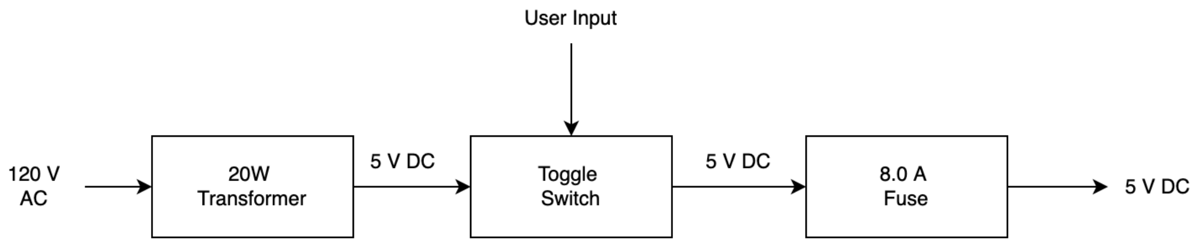


Figure 6: Power Subsystem

Software Sub-System

The software Sub-System primarily consists of a Raspberry Pi Zero as the main component. The processor takes in 5 Volts DC and a test switch and start-up button. The switch determines if the processor is in testing mode or regular mode and the startup button turns on Dormouse. The digital outputs will be used to power motors and LEDS. The PWM output will be used for the motors, and the 3.3 V will be used for any components that need to stay on.

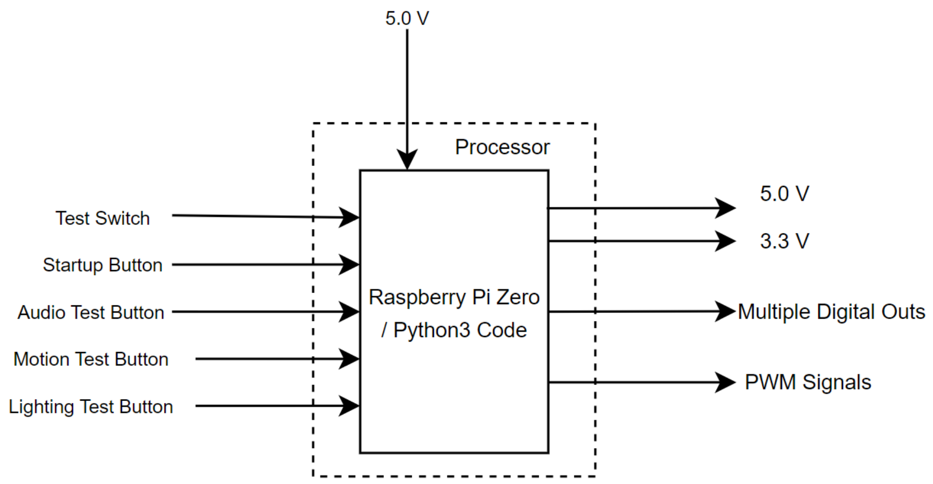


Figure 7: Software Subsystem

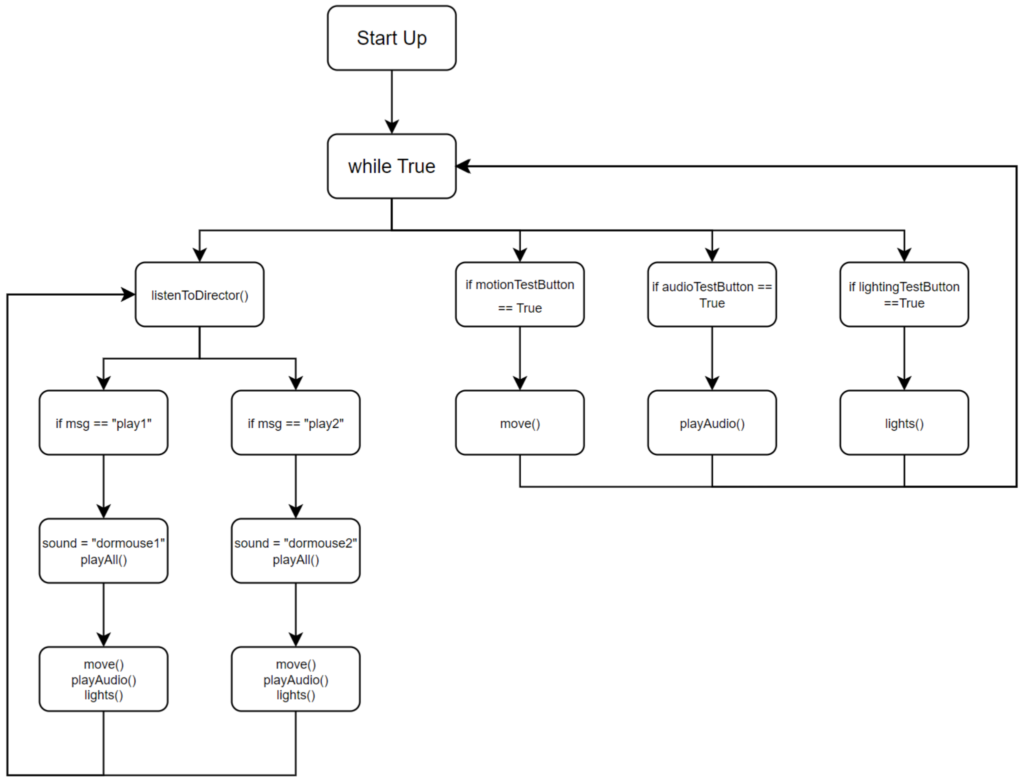

Start Up
while True
listenToDirector()
if motionTestButton == True
if audioTestButton == True
if lightingTestButton ==True
if msg == "play1"
if msg == "play2"
move()
playAudio()
lights()
sound = "dormouse1" playAll()
sound = "dormouse2" playAll()
move() playAudio() lights()
move() playAudio() lights()

Figure 8: Software Architecture

Software Design

Our current design begins by providing power to the system. From there, the system checks if the test switch is on. If the switch is on, Dormouse will respond to various test buttons. One button will play audio, the other will turn on the lights, and another will cause Dormouse to grow. If the switch is off, then Dormouse will connect to the director. Once connected, he will wait for the file needed for his performance. Once decoded, Dormouse will synchronize the commands with movement, lighting, and audio. Once done, Dormouse will power off.

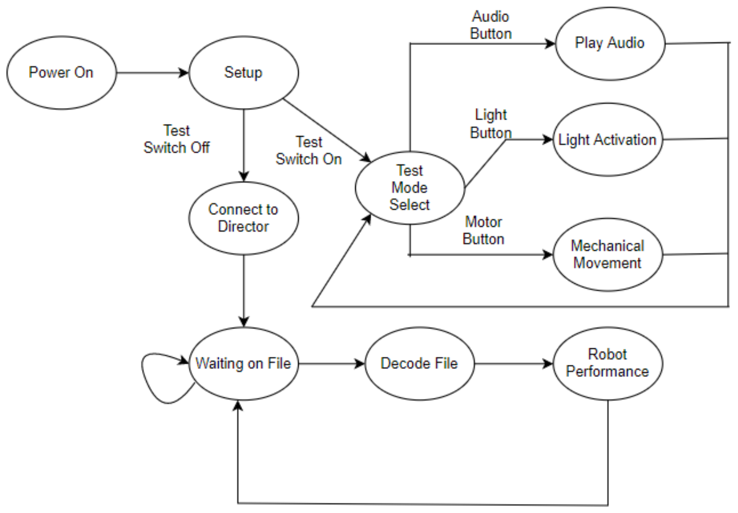
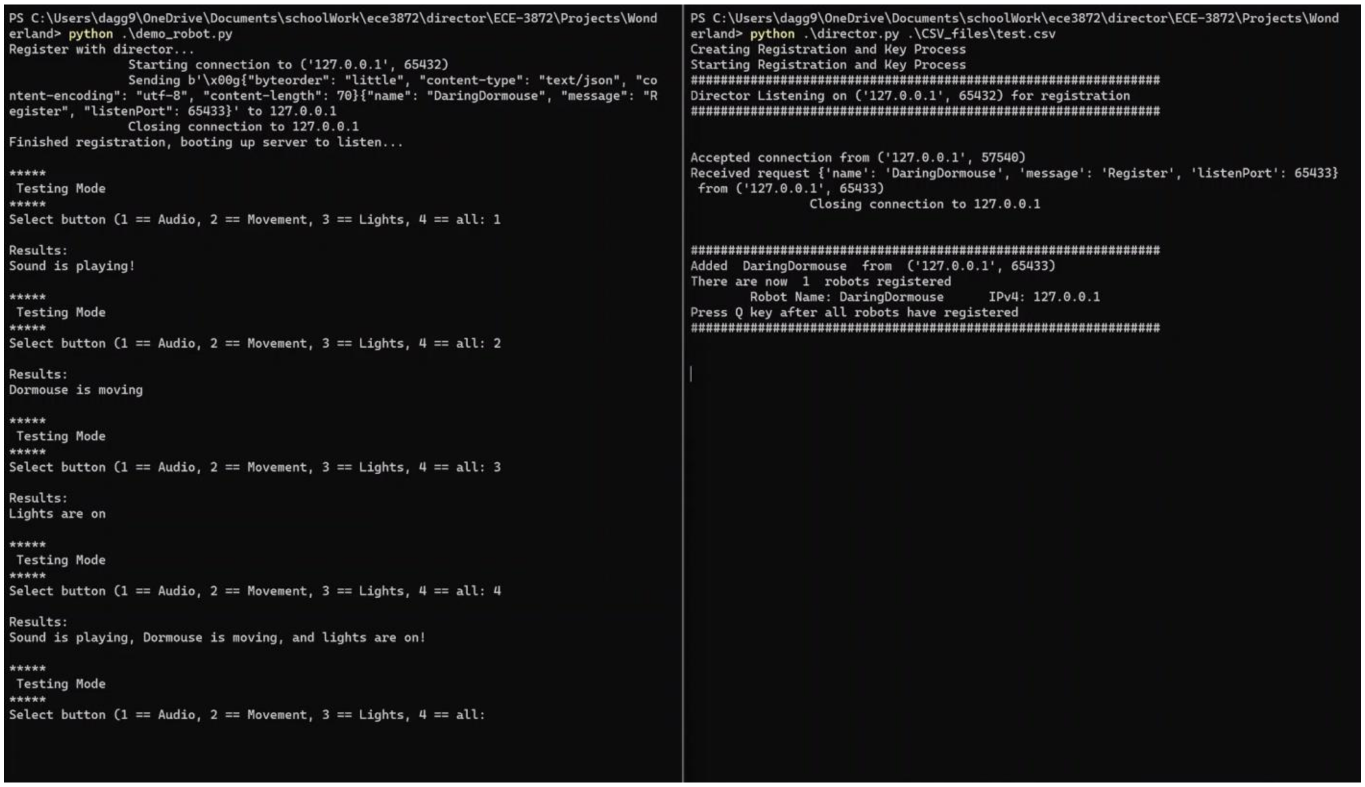


Figure 9: Software state machine with various states.

To simulate our software system, we ran the director and test robot code locally on the same machine. Instead of having a physical test switch, we used a variable to simulate it. When the variable was set to a value of one, the test robot would enter testing mode. When the variable was set to a value of zero, then the robot would wait to connect to the director.

When the switch variable was enabled and the robot entered testing mode, the robot lets the user know it is in testing mode via print statements. Next the user is asked to select a button. To simulate the buttons, the user is asked to enter a numerical value. Entering “1” results in the message “Sound is playing!” being shown on the screen. Entering “2” results in the message “Dormouse is moving” being shown on the screen. Entering “3” results in the message “Lights are on” being shown on the screen. Entering “4” simulates all three actions simultaneously. The message “Sound is playing, Dormouse is moving, and lights are on!” being shown on the screen.



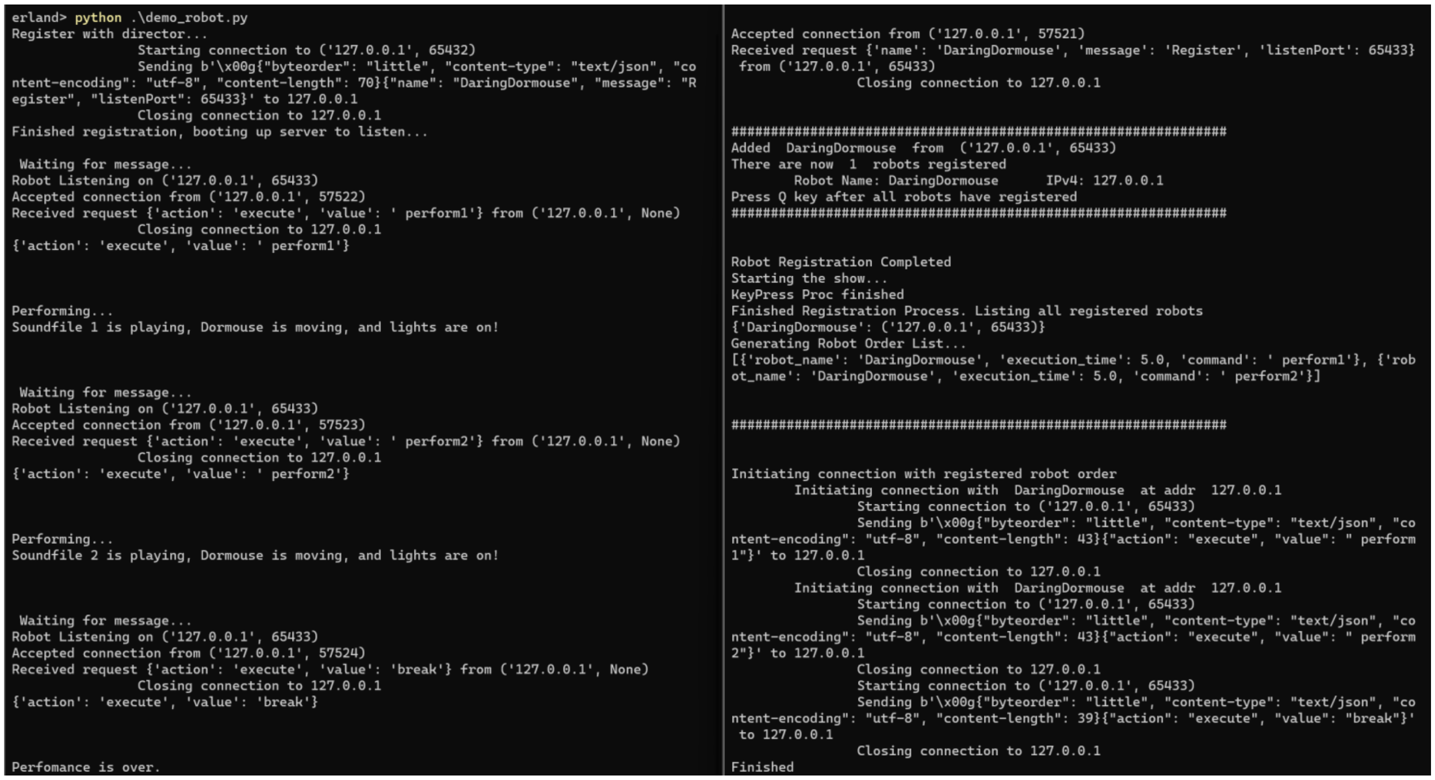
```
PS C:\Users\dagg9\OneDrive\Documents\schoolWork\ece3872\director\ECE-3872\Projects\Wond
erland> python .\demo_robot.py
Register with director...
                Starting connection to ('127.0.0.1', 65432)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/json", "co
ntent-encoding": "utf-8", "content-length": 70}{"name": "DaringDormouse", "message": "R
egister", "listenPort": 65433}' to 127.0.0.1
                Closing connection to 127.0.0.1
Finished registration, booting up server to listen...

*****
 Testing Mode
*****
Select button (1 == Audio, 2 == Movement, 3 == Lights, 4 == all: 1

Results:
Sound is playing!

*****
 Testing Mode
*****
Select button (1 == Audio, 2 == Movement, 3 == Lights, 4 == all: 2

Results:
Dormouse is moving

*****
 Testing Mode
*****
Select button (1 == Audio, 2 == Movement, 3 == Lights, 4 == all: 3

Results:
Lights are on

*****
 Testing Mode
*****
Select button (1 == Audio, 2 == Movement, 3 == Lights, 4 == all: 4

Results:
Sound is playing, Dormouse is moving, and lights are on!

*****
 Testing Mode
*****
Select button (1 == Audio, 2 == Movement, 3 == Lights, 4 == all:
```

```
PS C:\Users\dagg9\OneDrive\Documents\schoolWork\ece3872\director\ECE-3872\Projects\Wond
erland> python .\director.py .\CSV_files\test.csv
Creating Registration and Key Process
Starting Registration and Key Process
##############################################################
Director Listening on ('127.0.0.1', 65432) for registration
##############################################################


Accepted connection from ('127.0.0.1', 57540)
Received request {'name': 'DaringDormouse', 'message': 'Register', 'listenPort': 65433}
 from ('127.0.0.1', 65433)
                Closing connection to 127.0.0.1


##############################################################
Added  DaringDormouse  from  ('127.0.0.1', 65433)
There are now  1  robots registered
        Robot Name: DaringDormouse      IPv4: 127.0.0.1
Press Q key after all robots have registered
##############################################################
```

Figure 10: Software Simulation of Test Mode. Robot is on the left; Director is on the right.

When the switch variable was disabled and the robot ran as normal, the robot would connect to the director. Once connected, the user presses the "Q" key to begin the performance. The robot then waits for commands from the director, which are read from a CSV file. We tested our simulation with two commands, "perform1" and "perform2." When the command is given and read, the robot prints the message "Performing...."

Because we plan to use two different audio files, the robot printed one of two messages based on the command given. If "perform1" is given, the message "Soundfile 1 is playing, Dormouse is moving, and lights are on!" was printed. If "perform2" is given, the message "Soundfile 2 is playing, Dormouse is moving, and lights are on!" was printed. Once the director is finished giving commands, the message "Performance is over" is printed.



```
erland> python .\demo_robot.py
Register with director...
                Starting connection to ('127.0.0.1', 65432)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/json", "co
ntent-encoding": "utf-8", "content-length": 70}{"name": "DaringDormouse", "message": "R
egister", "listenPort": 65433}' to 127.0.0.1
                Closing connection to 127.0.0.1
Finished registration, booting up server to listen...

 Waiting for message...
Robot Listening on ('127.0.0.1', 65433)
Accepted connection from ('127.0.0.1', 57522)
Received request {'action': 'execute', 'value': ' perform1'} from ('127.0.0.1', None)
                Closing connection to 127.0.0.1
{'action': 'execute', 'value': ' perform1'}



Performing...
Soundfile 1 is playing, Dormouse is moving, and lights are on!



 Waiting for message...
Robot Listening on ('127.0.0.1', 65433)
Accepted connection from ('127.0.0.1', 57523)
Received request {'action': 'execute', 'value': ' perform2'} from ('127.0.0.1', None)
                Closing connection to 127.0.0.1
{'action': 'execute', 'value': ' perform2'}



Performing...
Soundfile 2 is playing, Dormouse is moving, and lights are on!



 Waiting for message...
Robot Listening on ('127.0.0.1', 65433)
Accepted connection from ('127.0.0.1', 57524)
Received request {'action': 'execute', 'value': 'break'} from ('127.0.0.1', None)
                Closing connection to 127.0.0.1
{'action': 'execute', 'value': 'break'}



Perfomance is over.
```

```

Accepted connection from ('127.0.0.1', 57521)
Received request {'name': 'DaringDormouse', 'message': 'Register', 'listenPort': 65433}
 from ('127.0.0.1', 65433)
                Closing connection to 127.0.0.1


###############################################################
Added  DaringDormouse  from  ('127.0.0.1', 65433)
There are now  1  robots registered
        Robot Name: DaringDormouse      IPv4: 127.0.0.1
Press Q key after all robots have registered
###############################################################


Robot Registration Completed
Starting the show...
KeyPress Proc finished
Finished Registration Process. Listing all registered robots
{'DaringDormouse': ('127.0.0.1', 65433)}
Generating Robot Order List...
[{'robot_name': 'DaringDormouse', 'execution_time': 5.0, 'command': ' perform1'}, {'rob
ot_name': 'DaringDormouse', 'execution_time': 5.0, 'command': ' perform2'}]


###############################################################


Initiating connection with registered robot order
        Initiating connection with  DaringDormouse  at addr  127.0.0.1
                Starting connection to ('127.0.0.1', 65433)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/json", "co
ntent-encoding": "utf-8", "content-length": 43}{"action": "execute", "value": " perform
1"}' to 127.0.0.1
                Closing connection to 127.0.0.1
        Initiating connection with  DaringDormouse  at addr  127.0.0.1
                Starting connection to ('127.0.0.1', 65433)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/json", "co
ntent-encoding": "utf-8", "content-length": 43}{"action": "execute", "value": " perform
2"}' to 127.0.0.1
                Closing connection to 127.0.0.1
                Starting connection to ('127.0.0.1', 65433)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/json", "co
ntent-encoding": "utf-8", "content-length": 39}{"action": "execute", "value": "break"}'
 to 127.0.0.1
                Closing connection to 127.0.0.1
Finished
```

Figure 11: Software Simulation of Robot running normally. Robot is on the left; Director is on the right.

Electronic Design

Our electric design is centered around a Raspberry 2 W, a custom-built printed circuit board, and two breakout boards that are attached to the printed circuit board. The figure below shows a high-level block diagram of the system.

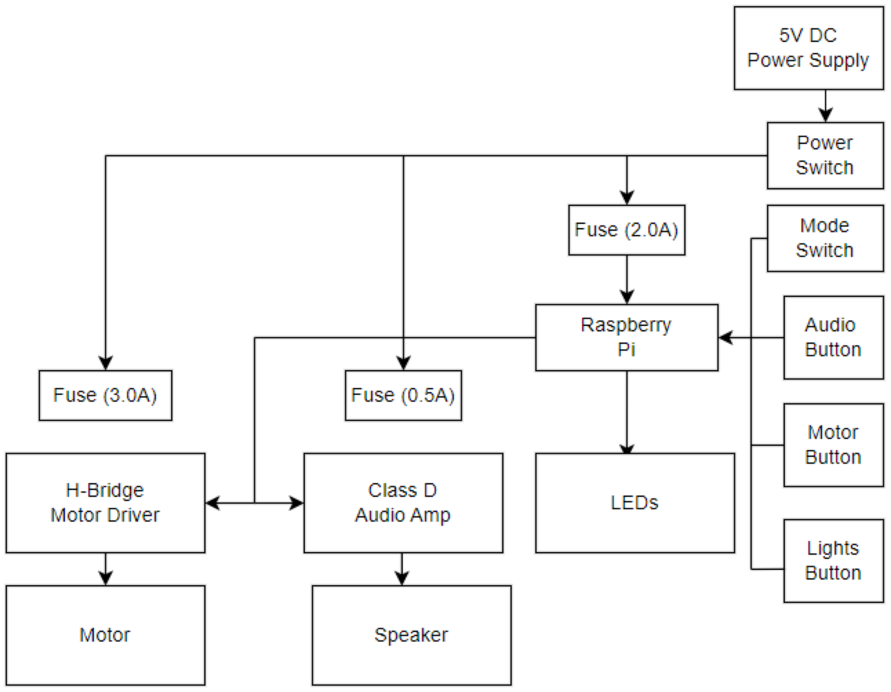


Figure 12: High-Level System Block Diagram of Electronic Design

After testing each subsystem circuit on a breadboard, we created a schematic using Eagle. The schematic allows for pinout to be assigned in and for proper components to be selected in our design.

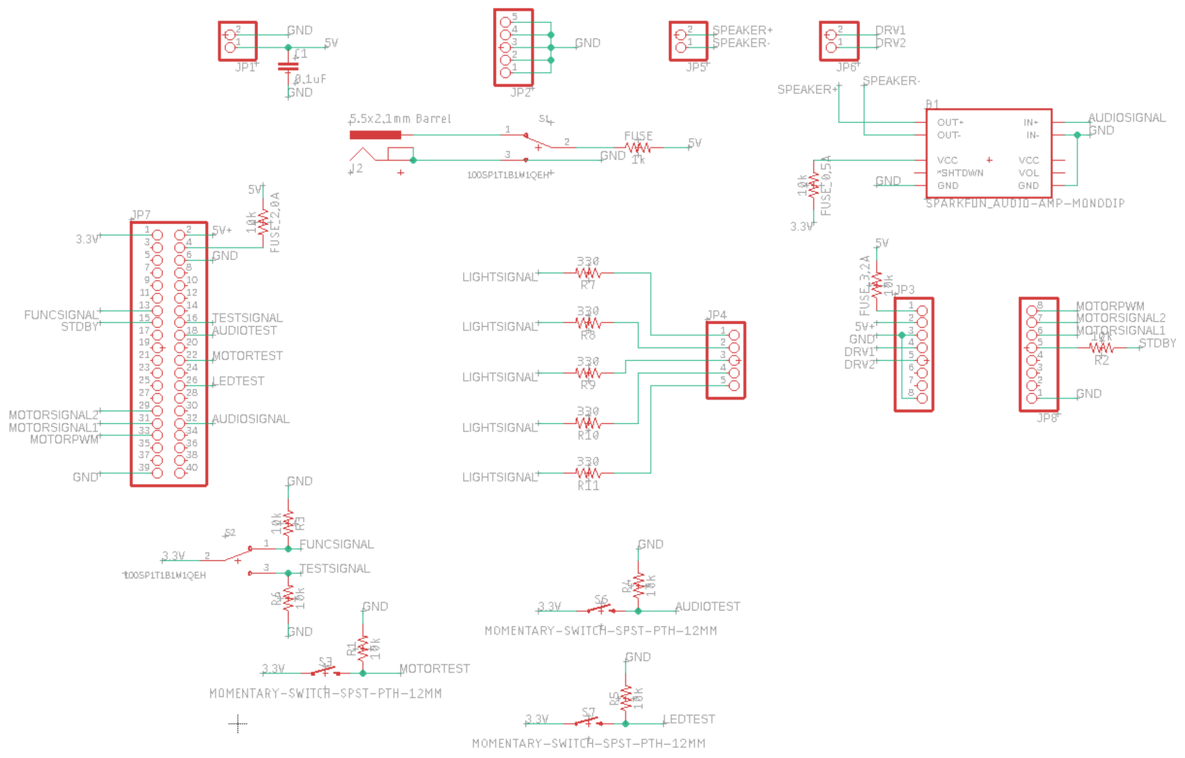


Figure 13: Initial Schematic created with Eagle. Includes switches, buttons, connectors for Audio amp, H-bridge driver and Raspberry Pi, speakers, power switch, and motor.

Once PCB layout was created and received, we began soldering components. Midway through the process we discovered an unintended connection between the power pin and input pin of the H-Bridge Driver breakout board. We made multiple attempts to solve the problem, but ultimately had to scrap a board and restart soldering. To prevent this problem from repeating, we changed the through-hole ports to connectors pins, that way we could remove components easily.

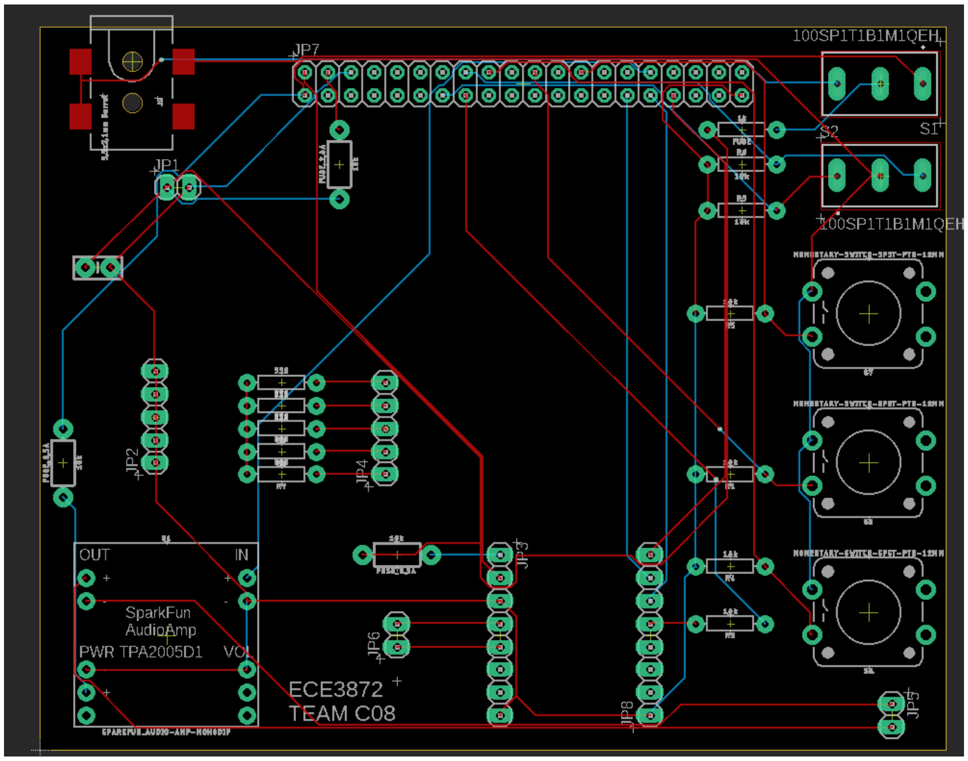


Figure 14: Printed Circuit Board created in Eagle.

Mechanical Design

Our mechanical design consists primarily of a scissor lift and teapot, both of which were 3D printed with Polylactic Acid (PLA). The scissor lift requires a total of six rods (circular cylinders) to connect two three-hole arm bars

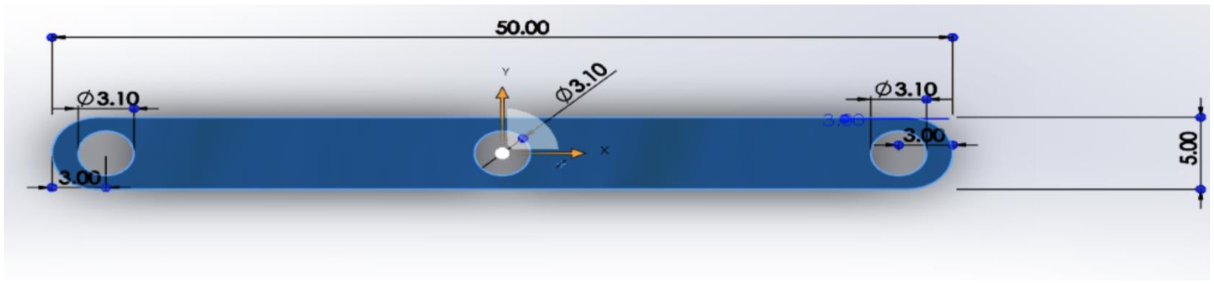


Figure 15: Arm bar design

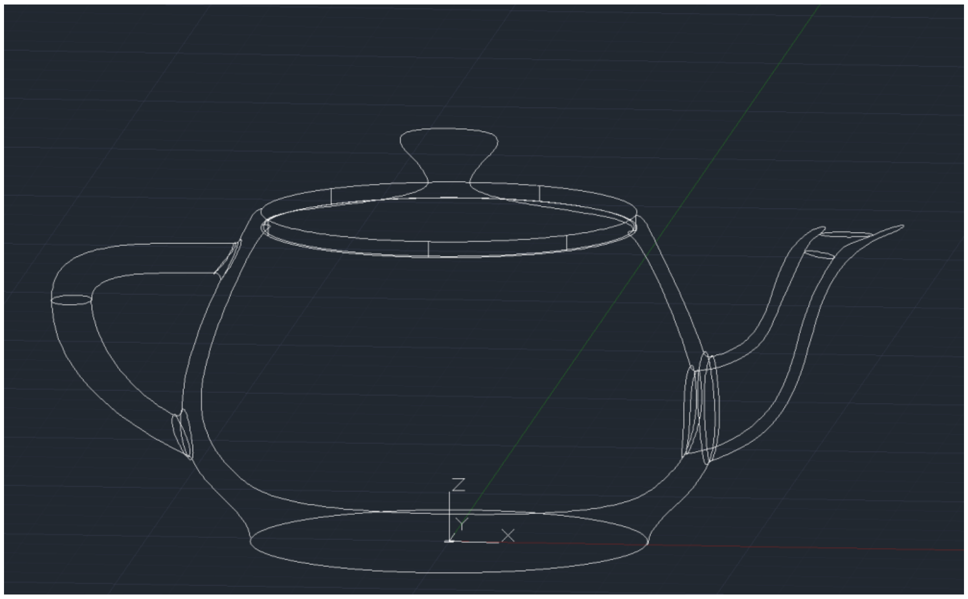


Figure 16: Teapot design

After printing and testing the fit of a variety of rod diameters in the armbar holes, we constructed the final product scissor lift.

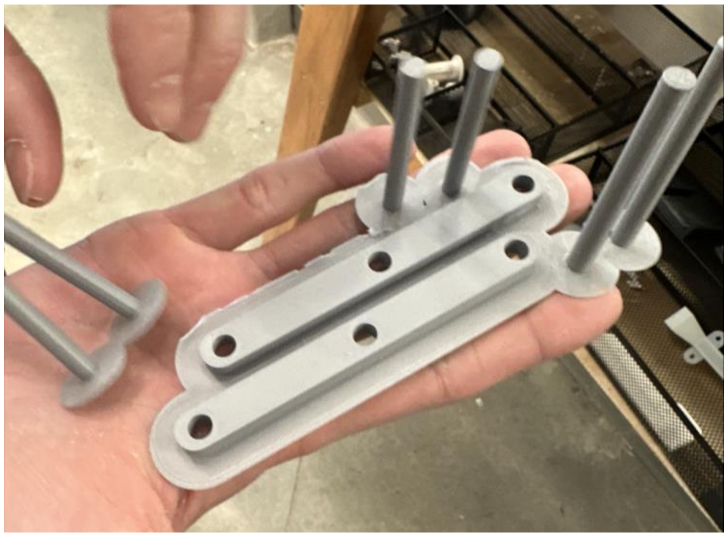


Figure 16: Arm bars and rods for scissor lift

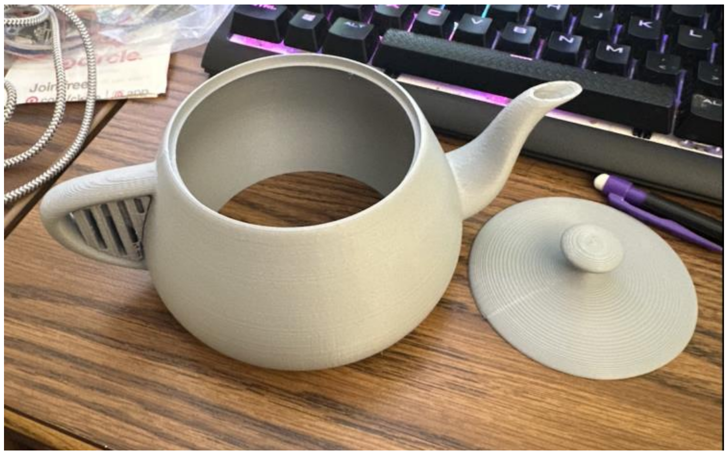


Figure 17: Teapot base and lid

This design works by opening (collapsing) and closing (extending) the lift with alternating rotation of the DC motor. We secured the armbar base closest to the motor and attached two pieces of string, one for each side of the motor shaft, to the opposite base. As the motor begins to rotate, the string gets taut, and the scissor lift raises Dormouse until the motor PWM signal changes. At this point the shafts rotate in the opposite direction to unwind the string, which allows for the weight of dormouse and the teapot lid to lower the lift.

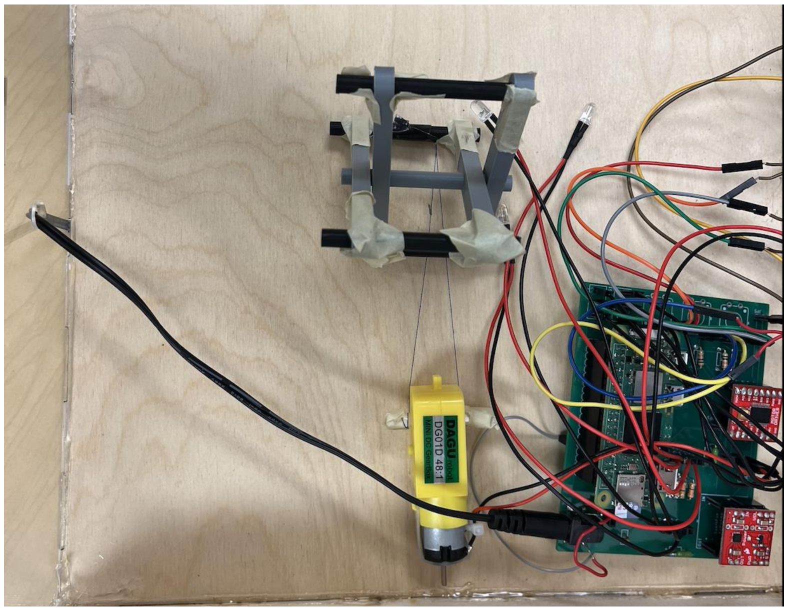


Figure 18: Scissor lift attached to 4.5 V DC motor

Integration Testing

Each subsystem needs to work alongside one another. To ensure they work properly, we did a few simulations.

We tested each subsystem individually before combing them. By doing everything separately, we ensured that minor mistakes can be found easily and be fixed. Had we just combined everything and began testing then, it would be difficult to find errors. This method was also more convenient for the team, as we all had busy schedules and were not required to be available at the same time.

We began by testing circuits on TinkerCAD. We tested circuits for our lights, speaker, and audio to ensure they would work. Once we knew they worked, we tested the same three circuits on a breadboard. All circuits were tested separately.

At the same time this was being tested, test software was being written. Since we were not ready to connect the circuits at the time, we instead printed descriptive statements.

Once we were confident with each subsystem, we tested them all on a breadboard successfully. Our next step was to add the mechanical aspects and test everything on the PCB (Printed Circuit Board).

Unfortunately, our PCB had plenty of issues that needed to be fixed first. Because of this setback, our integration testing was delayed by about a week. However, once these issues were fixed everything worked flawlessly.

Conclusion

Throughout this class, we finished with a successful project that met the necessary base requirements. The suggested method of using subsystems worked great for our team. The software subsystem ended up being the simplest one to set up, with any issues being quickly resolved. The electric subsystem was more difficult, as our PCB had plenty of issues. The team became familiar with debugging PCBs (printed circuit boards) by the time the project was finished. The mechanical subsystem also proved to have its challenges, but those were able to be resolved quickly.

If we were tasked with this project again, we would make quite a few changes. Our lighting was extremely limited. The lights were only one color and they all had to be turned on at the same time, limiting potential patterns. Instead of using LEDs lined in parallel, we could instead use Neopixels. Neopixels allow for RGB color, and each light can be set independently of each other.

Additionally, we would use a different movement system. The scissor lift design did work for our design but proved to be difficult to set up. Had we used a different system, we could have saved time and effort and placed more focus on other areas of our design. These changes are expensive, but our final budget would have allowed it.

Overall, we believe that we were successful with the project. We finished with a working robot, and we learned plenty of skills including soldering, circuit debugging, laser cutting, CAD, and PCB design. For most of us, this was the first time we were able to really utilize the resources available in the Hive. We met all the base requirements, and we are very proud of the work that has been done on the project.

Schedule

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

	Week Number												
Task	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Brainstorm	Dylan	Dylan											
Design Top Level Block Diagram	Dylan	Dylan											
Sketch Design	David	David											
Design Software Block Diagram	David	David											
Electrical Goals	Adam	Adam											
Mechanical Goals	Dylan	Dylan											
Software Goals	David	David											
Audio Goals	Adam	Adam											
Visual Goals	David	David											
Establish Requirements	Adam	Adam											
Complete Proposal	Adam	Adam											
Proposal		⬛											
Software Architecture		David	David										
Software Design		David	David										
Electrical Circuit Design		Adam	Adam										
Electrical Circuit Simulation		Adam	Adam										
Mechanical Drawings		Dylan	Dylan										
Lighting Design		Dylan	Dylan										
Motion Design		Dylan	Dylan										
Power Design		Adam	Adam										
Audio Design		Dylan	Dylan										
Finalize Visual Design		Adam	Adam										
Audio Circuit Design				Adam	Adam								
Software Simulation				David	David								
Complete PDR				Dylan	Dylan								
PDR					⬛								
Software mechanical code build						David	David						
Software lighting code build						David	David						
Software audio code build						David	David						
Electrical Component Selection					Adam	Adam	Adam						
Electrical PCB Modeling						Dylan	Dylan						
Order of Electrical board							Dylan						
Mechanical Modeling with Visual look					Dylan	Dylan	Dylan						
Electrical Circuit Simulation							Adam	Adam					
Audio Simulation						Dylan	Dylan						
Create Testing Requirement Documents					Dylan	Dylan	Dylan						
Prepare BOM						Dylan	Dylan	Dylan					
Prepare CDR							David	David					
CDR								⬛					
Software Build								David	David				
Electrical Build								Adam	Adam				
Power Build							Adam	Adam					
Mechanical Build										Dylan	Dylan		
Software audio test									David	David	David		
Software electrical test									David	David	David		
Software mechanical test									David	David	David		
Lighting Test										Adam	Adam		
Motion/Mechanical Test									Dylan	Dylan	Dylan	Dylan	
Power Test									Adam	Adam			
Software Debug									David	David	David	David	
Electrical PCB testing										Adam	Adam		
Audio Test									Adam	Adam	Adam		
System Integration										Dylan	Dylan		
System Test										David	David	David	
Prepare Final Presentation										Adam	Adam	Adam	
Compile final documentation										David	David	David	David
Final Inspection and Demonstration												⬛	
Milestones	David												
Tasks	Adam												
	Dylan												

Table 2: Schedule to complete Dormouse