Final Design Document

MAD HATTER In The Hat

Team #10B: Rushabh Shah, Zachary Crawford, Auveed Rokhsaz, Matthew Hannay

Date: November 9, 2022

Table of Contents

Revision Record

Date	Author	Comments
Sept 9, 2022	Team	Document Created (system design and hierarchical design)
Sept 9, 2022	Rushabh and Zachary	Changed the title, date, added members
Sept 16, 2022	Rushabh and Matthew	Wrote and changed figure for the Project Description, wrote and changed figure and table for System Design, and wrote the Sub System Design, wrote and changed figures for Mechanical Design
Sept 21, 2022	Rushabh	Updated Project Description, System Design, User Interface Sub System, Audio Sub System
Sept 23, 2022	Rushabh and Auveed	Changed figure for Sub System Design, Motion Sub System, updated Motion Sub System, Power Subsystem, Processor Subsystem, Mechanical Design, Software Design
Sept 24, 2022	Rushabh	Updated Project Description, System Design, Audio Subsystem, Motion Subsystem, Power Subsystem, Processor Subsystem, Constraints section, and Electronic Subsystem Updated figure for Power Subsystem, Electronic Subsystem Added Light Subsystem
Sept 25, 2022	Team	Finalized PDR
Oct 11, 2022	Rushabh	Created the CDR
Oct 13, 2022	Rushabh	Updated the CDR
Oct 14, 2022	Rushabh, Matthew, Auveed	Electrical, software, and mechanical figures were updated. Added software simulations and design flow.
Oct 15, 2022	Rushabh	Added a component section for all subsystems
Oct 16, 2022	Rushabh	Fixed schedule
Nov 5, 2022	Rushabh	Updated and fixed the schedule
Nov 9, 2022	Rushabh	Created the Final Document
Nov 11, 2022	Team	Updated and finalized the document

Project Description

We have designed a box with a Mad Hatter figure in the center of a rotating circular table that contains teacups on each side of the Mad Hatter. The box will be Georgia Tech-themed. In our design, Mad Hatter's Hat can move up and down according to our inputs which will sit on top of Mad Hatter. The light on the side of the rotating round table will turn on a certain way when the hat rises and another way when the hat drops. The potentiometer will increase/ decrease the loudness of the dialogue when the audio button gets pressed.

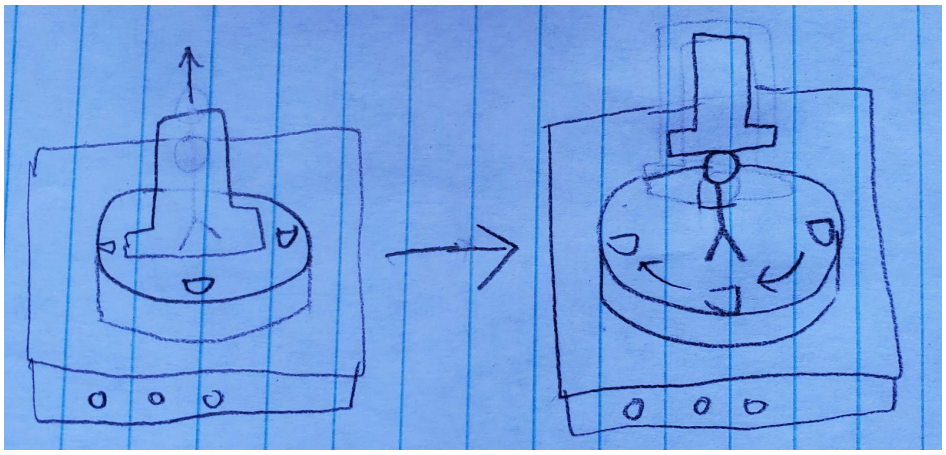

Figure 1: Our initial rendering of the Mad Hatter thespian

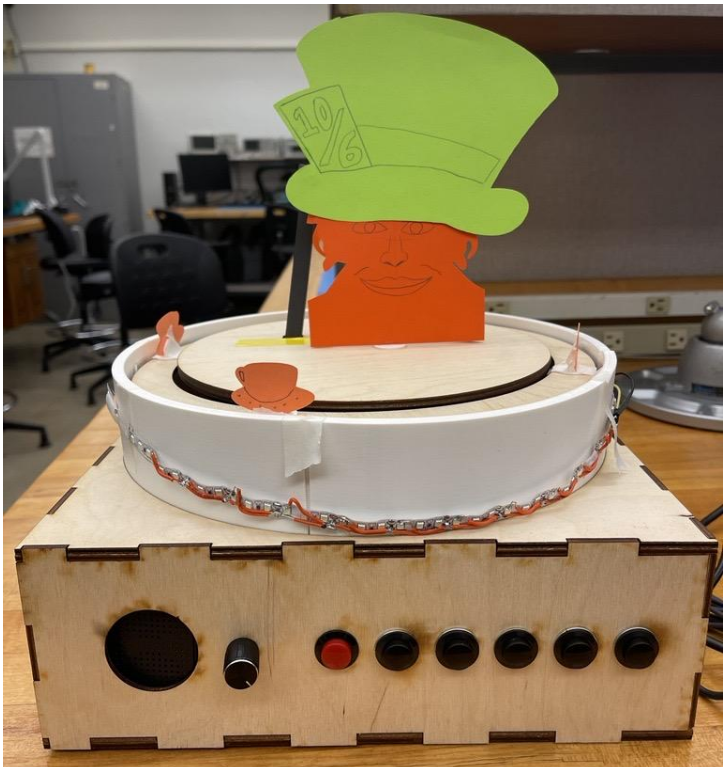

Figure 2: Our final rendering of the Mad Hatter thespian

System Design

The system's inputs include 120V AC input power, an analog audio source, and user input. The outputs of this system include a speaker, the movement produced by the figurine driven by servo motors, and the light demonstration activated after movement or dialogue. The system comprises six subsystems: the user interface, processor, power, motion, audio, and light. The user interface includes the reset, motion, light, audio, and all live push buttons. The processor consists of a Raspberry Pi processor and a Mbed processor. The audio sub-system comprises an audio jack, a speaker, a potentiometer to change the volume, and a mBed to compute the speaker volume using an analog input to the potentiometer. The motion system includes two servos, (0 to 180) and (0 to 270). The power sub-system includes a 120V AC to 5V DC transformer, a user input switch, and a fuse. The light subsystem contains an LED strip controlled by a GPIO pin from the Pi.

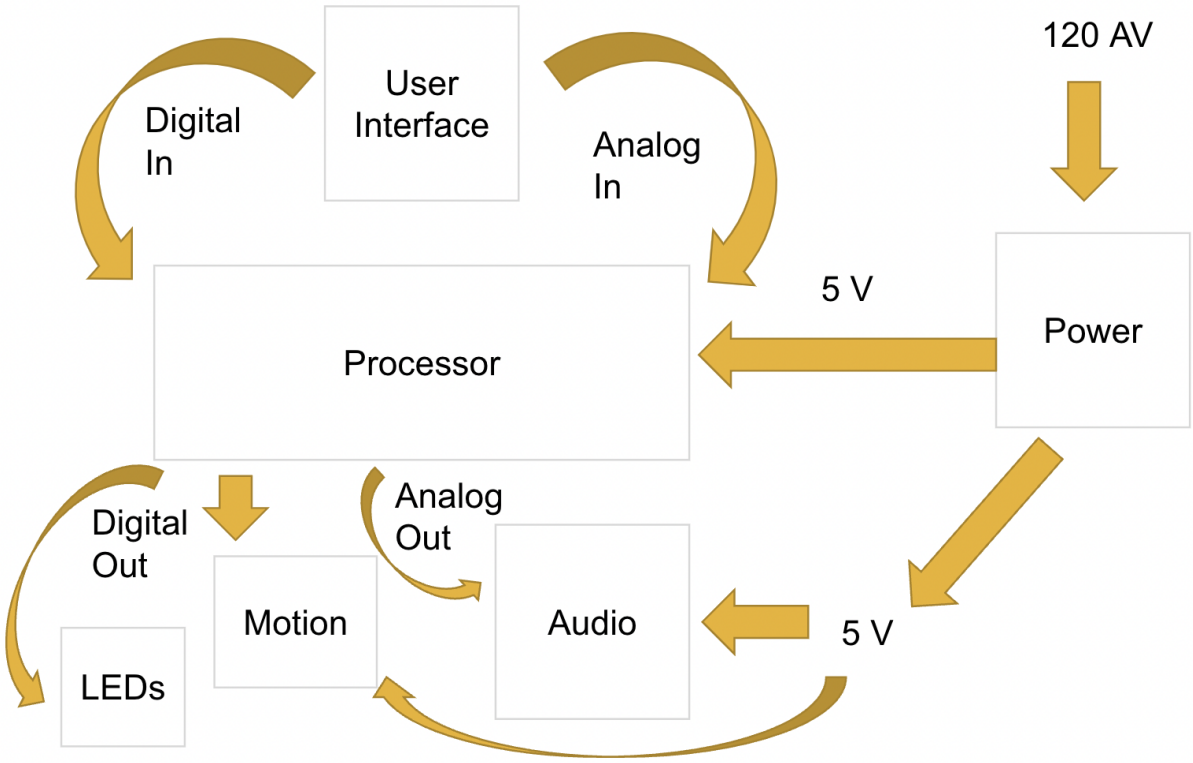

Figure 3: System Diagram

Table 1: Sub-system inputs and outputs

INTERFACE	SOURCE	DESTINATION	DESCRIPTION
			Analog signal between 0 V and 5V DC
Analog Tones	User Interface	Processor	Analog signal from the microphone
Digital Controls	User Interface	Processor	5V DC signal for the 5 push buttons
Servos Control Signals	Processor	Motion	PWM signals to control the servos
PWM Audio	Processor	Audio	PWM signal to output dialogues
LED	Processor	Light	PWM signal to output controlled brightness
120 V	System Input	Power	120 AC input power
		Processor	5V DC power 25 W
		Motion	5V DC power 15 W
5 V	Power	Audio	5V DC power 10 W

Sub-System Designs

Our design incorporates six sub-systems: user interface, computing, motion, power, audio, and light sub-systems. The audio subsystem consists of a potentiometer to control the output volume and a USB audio speaker connected to a USB output on the pi, the motion will execute the movement of the figure, the power will convert 120V AC to 5V DC, and the lights will turn on/off. The computing sub-system will integrate all of the sub-systems by converting the user input to signals for the audio, movement, and light systems.

User Interface Sub-System (Lead: Rushabh)

The user interface will take 5 Volts DC for the voltage divider circuit and will use an analog line IN for a test signal input in addition to the physical user input. This system will output analog signals from the analog IN user input button and the audio jack. It will output a digital signal from the four state buttons.

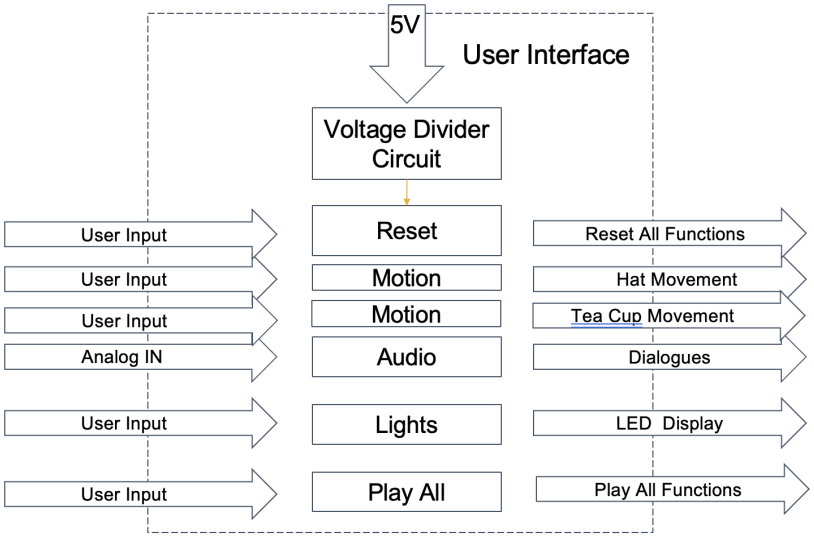

Figure 4: User Interface Sub-system

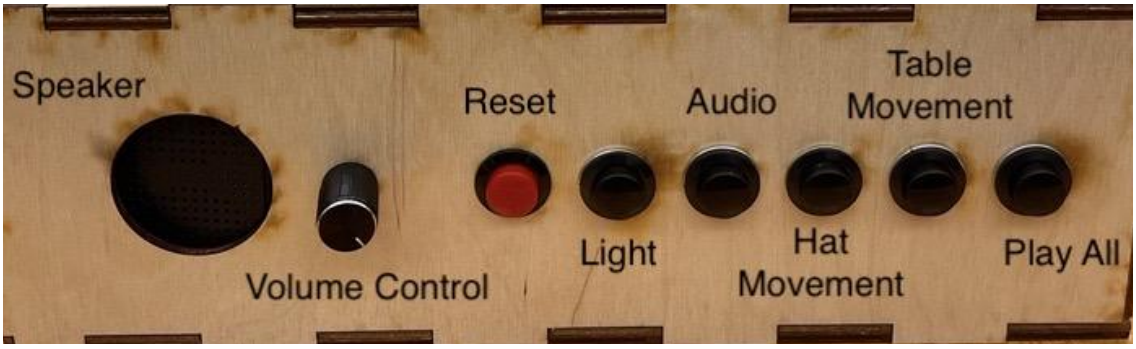

Figure 5: User Interface Sub-system for the actual Mad Hatter thespian

Audio Sub-System (Lead: AUVEED)

The audio system will take 5 Volts DC from the Power supply Module and sample an analog signal from the MBED processor using the potentiometer using the Pin 20 Analog In feature. The mBed will then convert the voltage to a float between zero and one representing the desired volume percentage. This float is then sent to the Raspberry Pi 4 via a USB serial port. The Pi then adjusts the volume of the audio output. The audio files are stored in a *.wav* format in the Pi's memory and are loaded as necessary by the software state machine system (part of the computing subsystem). The audio files are played through the Pi's USB port into the USB speaker.

Audio Components

- Audio Pushbutton
- Potentiometer
- Mbed processor
- Raspberry Pi processor
- USB Mini Stereo USB Speaker

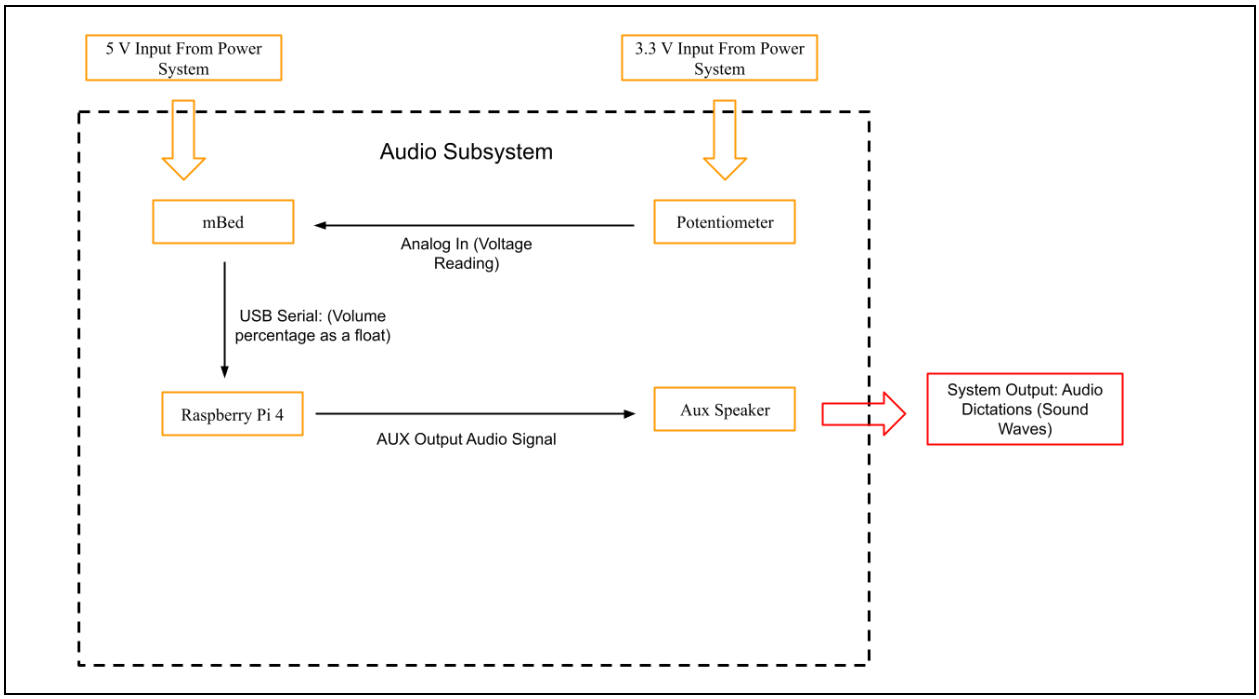

Figure 6: Audio Sub-System

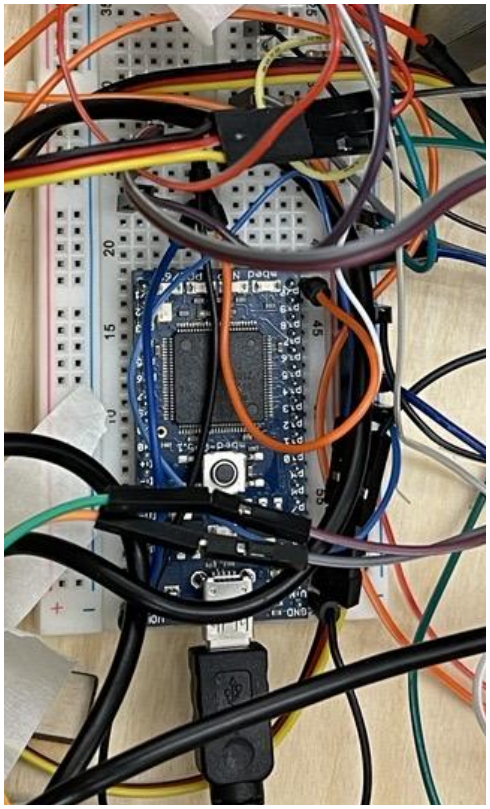

Figure 7: Audio Sub-System for the actual Mad Hatter thespian

Motion Sub-System (Lead: MATTHEW)

The motion subsystem will take 5 Volts DC from the Power supply Module and receive PWM Signal to control the servo motors from Raspberry Pi. The servo motors will then interpret the signals and power the motors appropriately to produce the desired motion.

Motion Components

- Motion Pushbutton
- Raspberry Pi processor
- 2 Servo motors

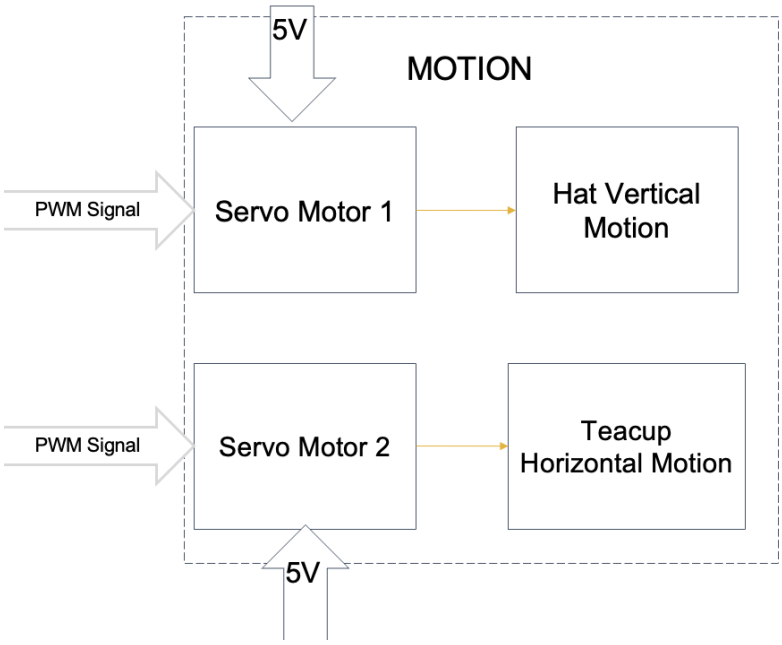

Figure 8: Motion Sub-System

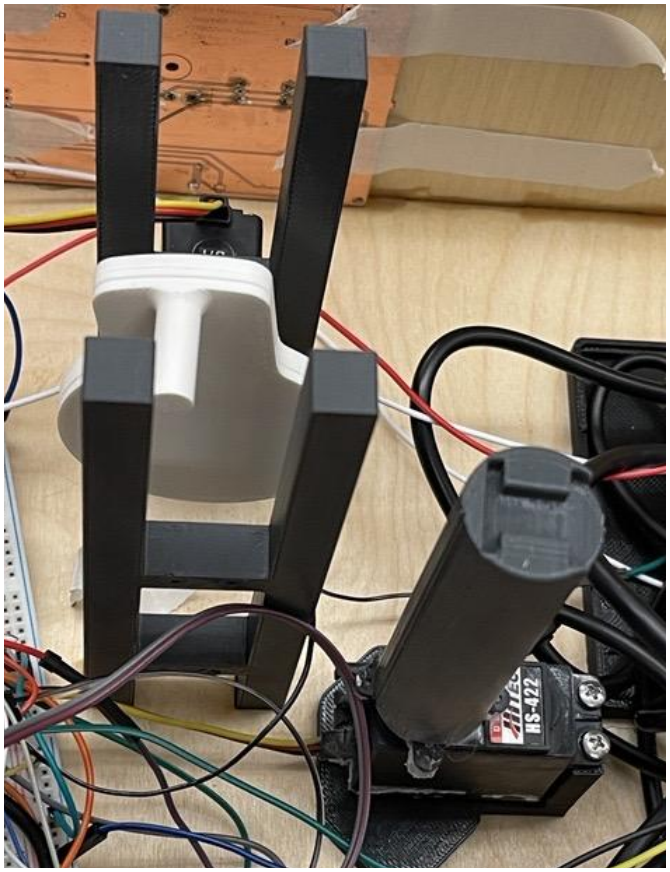

Figure 9: Motion Sub-System for the actual Mad Hatter thespian

Light Sub-System (Lead: Zachary)

The light subsystem will receive a GPIO output signal from Raspberry Pi and display various lit functionalities on the LEDs on the rotating table. The timing of the LEDs turning on is determined by the state machine in the computing subsystem.

Light Components

- Light Pushbutton
- Raspberry Pi processor
- LED Strip

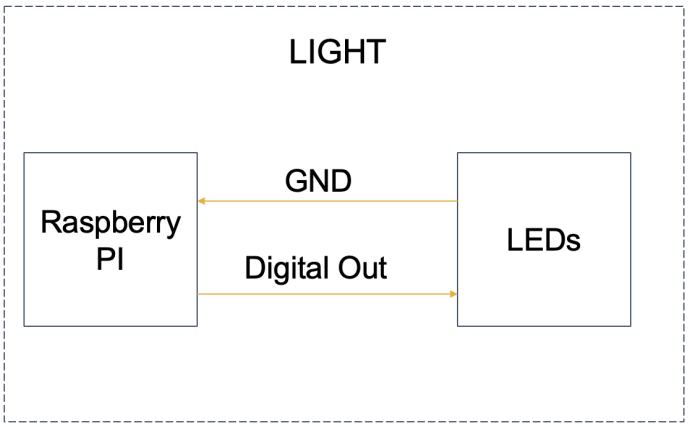

Figure 10: Light Sub-System

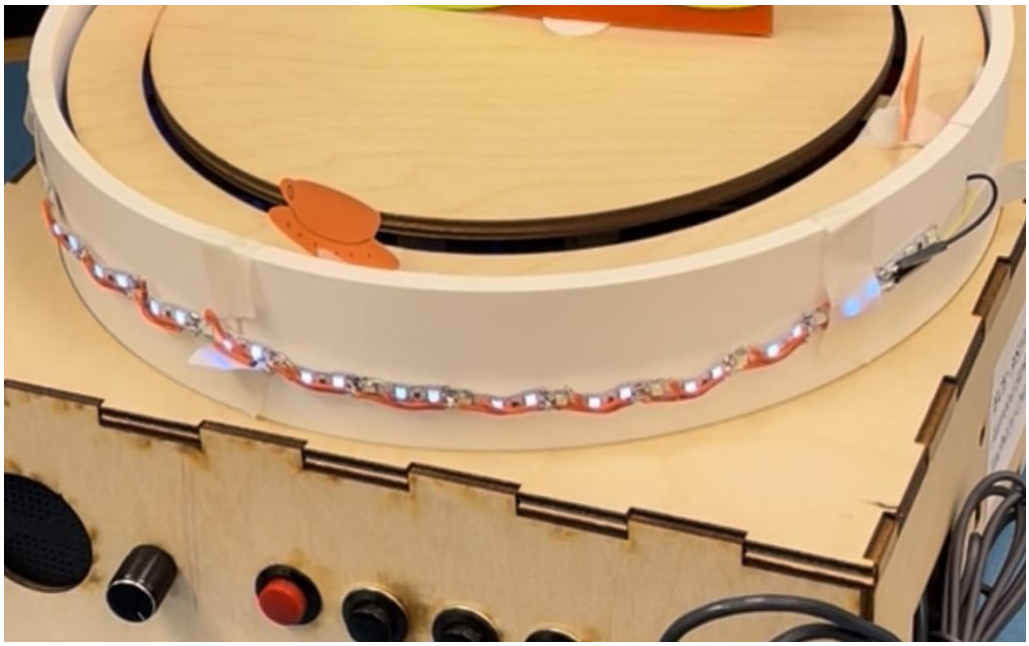

Figure 11: Light Sub-System for the actual Mad Hatter thespian

Power Sub-System (Lead: Rushabh)

The power subsystem will take in 120 Volts AC as an input and output 5 Volts DC. It will route power through a 20-Watt transformer, then through a power supply module to output the 5V DC to the Raspberry Pi, Mbed, and the PCB electronic components to send and receive the signals.

Power Components

- 120V AC
- 20 W Transformer
- 5V Power Supply Module
- Mini Power Switch

- Power Fuse

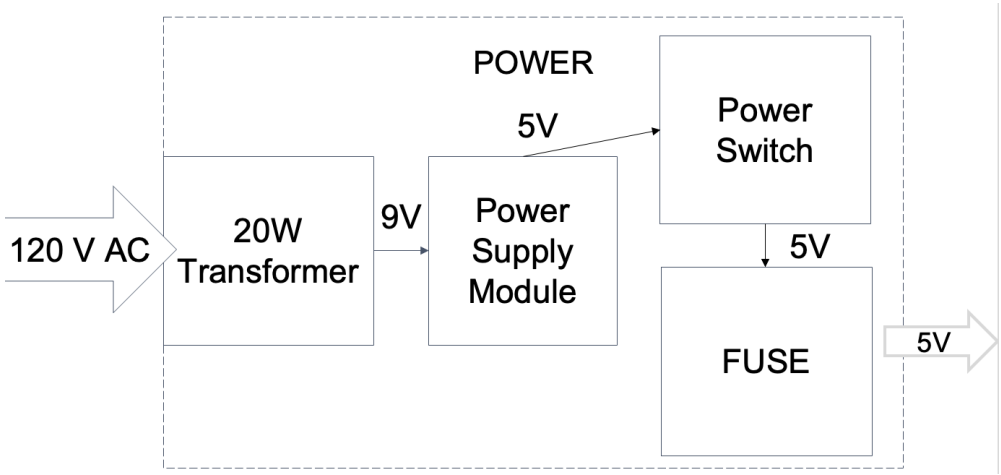

Figure 12: Power Sub-System

Figure 13: Power Sub-System for the actual Mad Hatter thespian

Computing Sub-System (Lead: Auveed Rokhsaz)

The computing sub-system will take 5 Volts DC in addition to analog inputs from the potentiometer and push buttons and will output PWM signals for the LEDs, digital signals for the motor, and USB signal for the audio played through the speaker. The computing sub-system will also output a 3.3V DC for the potentiometer.

Computing Components

- Potentiometer for analog in
- mBed processor
- USB Serial connection
- Pushbuttons
- Raspberry Pi processor
- LED strip
- Servo motors
- USB Mini Stereo Speaker

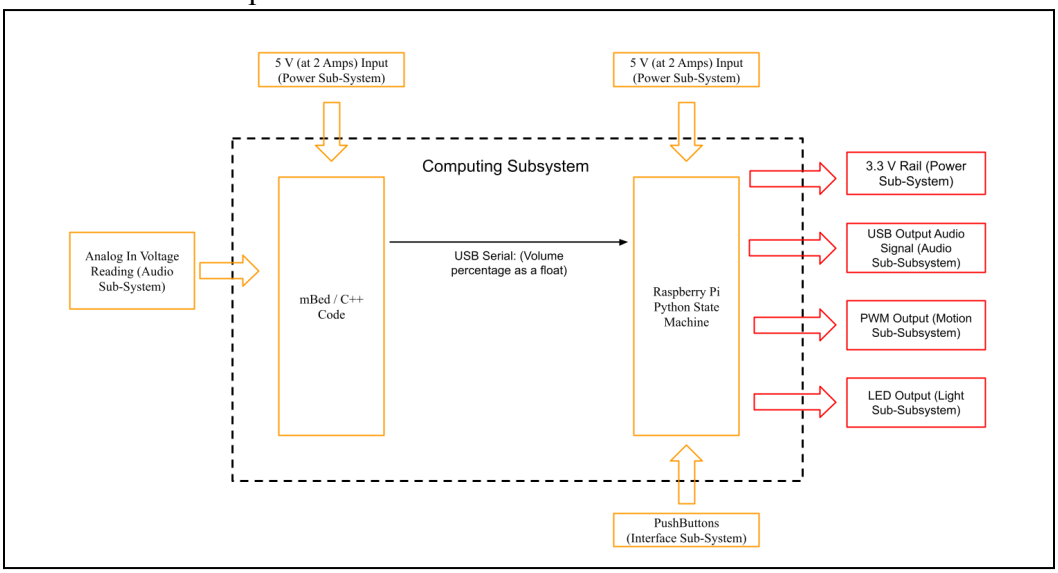

Figure 14: Processor Sub-System

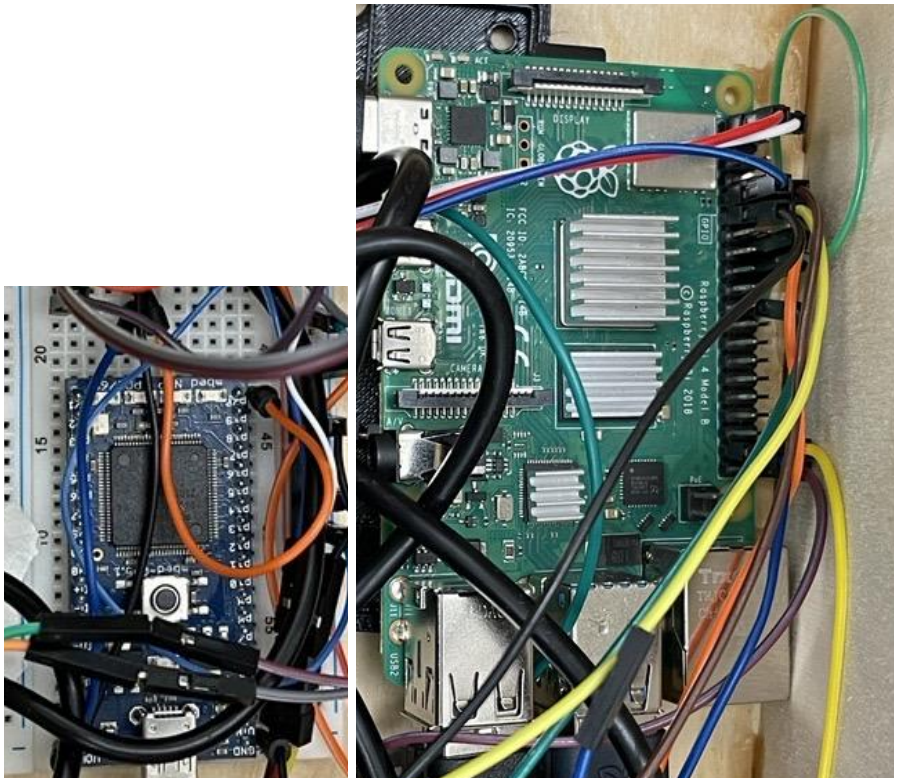

Figure 15: Processor Sub-System using Mbed and Raspberry Pi for the actual Mad Hatter thespian with its connections

Constraints, Alternatives, and Trade-Offs

One constraint with the servos was that we had to get the correct voltage so the servos and power supplies could be used together.

Motors: Servo Motor

- The servo motor was picked for the rotational motor because it is most accurate for the type of motion
- One servo motor will be used to vertically raise the hat enough to show the Mad Hatter. This motor will rotate until the hat reaches Mad Hatter's headline.
- Another servo motor will be used horizontally to rotate the table around the Mad Hatter. With enough power and motor rotation, we will be able to have a smooth rotation for the table.
- If adjustments are needed we will reevaluate our motor requirements.

One alternative with the audio subsystem was a speaker driver with a Class D amplifier. The Class D amplifier speaker system would have taken longer to get through to the speaker using the software required.

- The USB jack speaker was picked because it is easier to use from the viewpoint of software.

- We also added another processor, Mbed, to control the volume of the speaker using a potentiometer with an Analog In signal.

Budget is another constraint. Staying under $160 may be difficult as the design process begins and problems arise.

Electronic Design

Our electrical design is centered around a Raspberry Pi processor, a Mbed processor, a custom-built PCB, and several specialized breakout boards. The figure below shows a high-level block diagram of our system. The component diagram includes all inputs such as buttons and a power supply. Each of these inputs leads to the microcontrollers, for us a Raspberry Pi and Mbed, and out as an output through the servo motors, the speaker, and the LEDs.

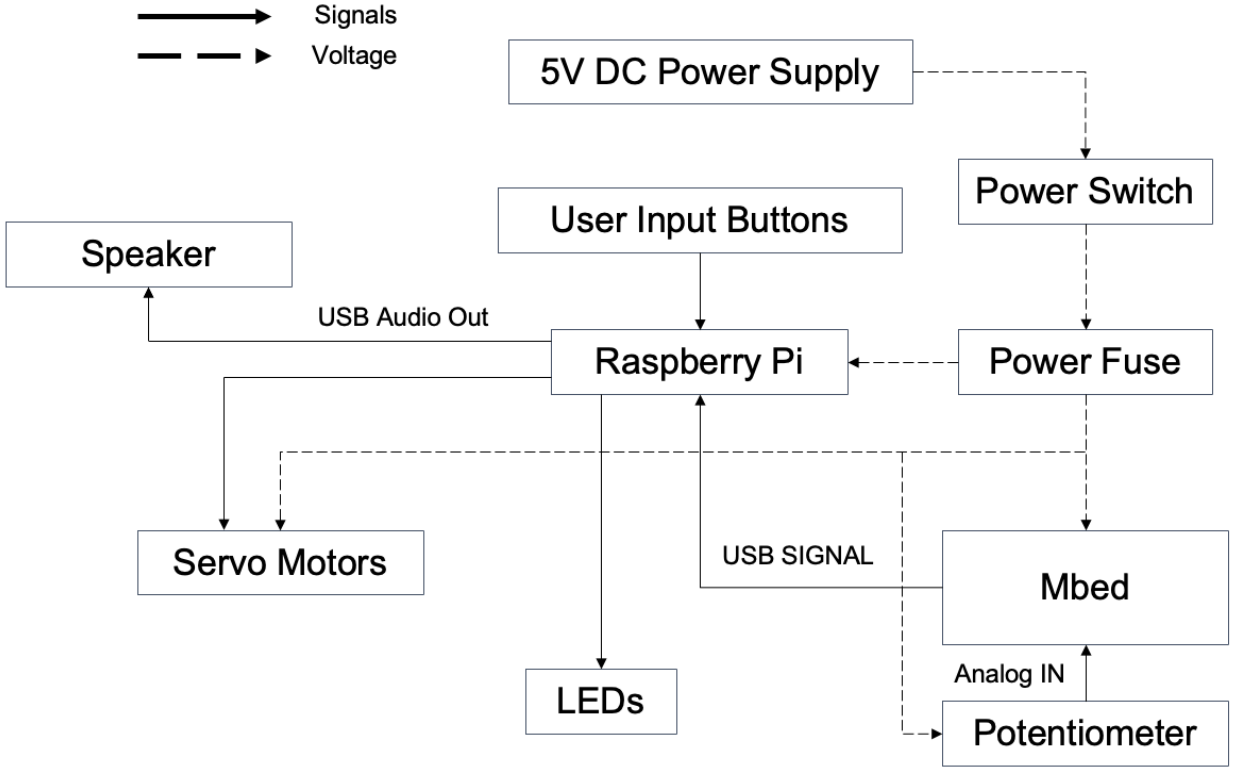

Figure 16: Electronic component block diagram

Electronic Testing and Integration

A series of simulations and tests have been designed to verify that each electronic component is functional separately during the development of the system with respect to the hierarchy shown in Figure 16 according to the software written based on the requirements we need to meet. Firstly, a very simple test using a breadboard and pushbuttons has been used to simulate the buttons that change states using the digital IN and analog IN. This test was simply tested to ensure that the pushbuttons work based on the output information from the console.

Similarly, the servo motors and the LEDs have been tested and checked based on their outputs using the digital out signals and the speaker has been tested and checked based on the outputs using the analog out signal in the USB port. After each component was tested and checked for, we then tested the buttons to their corresponding subsystem along with the reset button. In the end, we will integrate all the subsystems together for the Play All live button.

Created in Eagle, our schematic takes all necessary components from the block diagram and creates a pinout for each individual part. We tested all individual parts on a breadboard and then moved to our PCB.

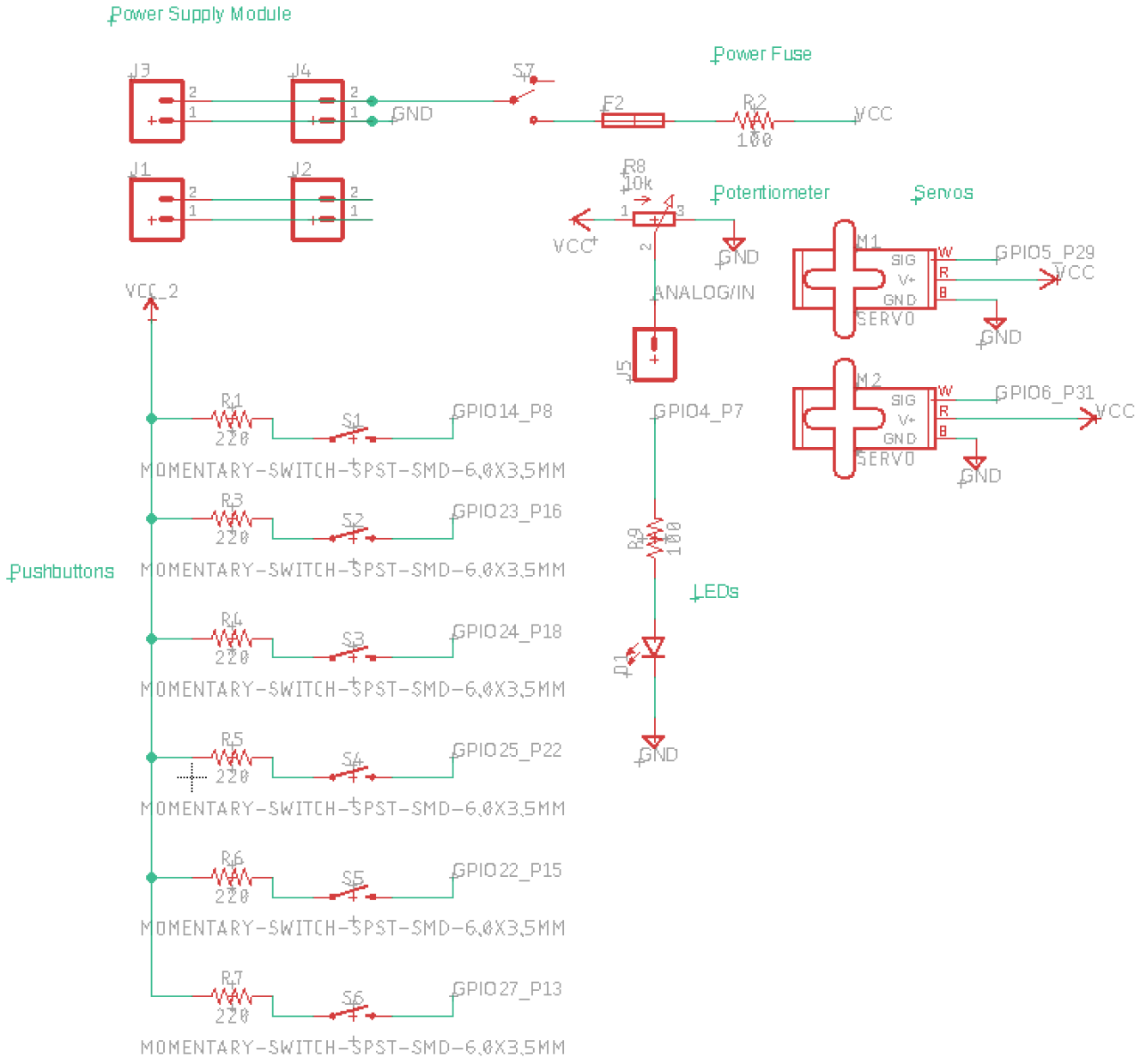

Figure 17: Our initial schematic created in Eagle that integrates Raspberry PI, Mbed, pushbuttons, a potentiometer for the speaker, and servos. PCB is designed from this schematic.

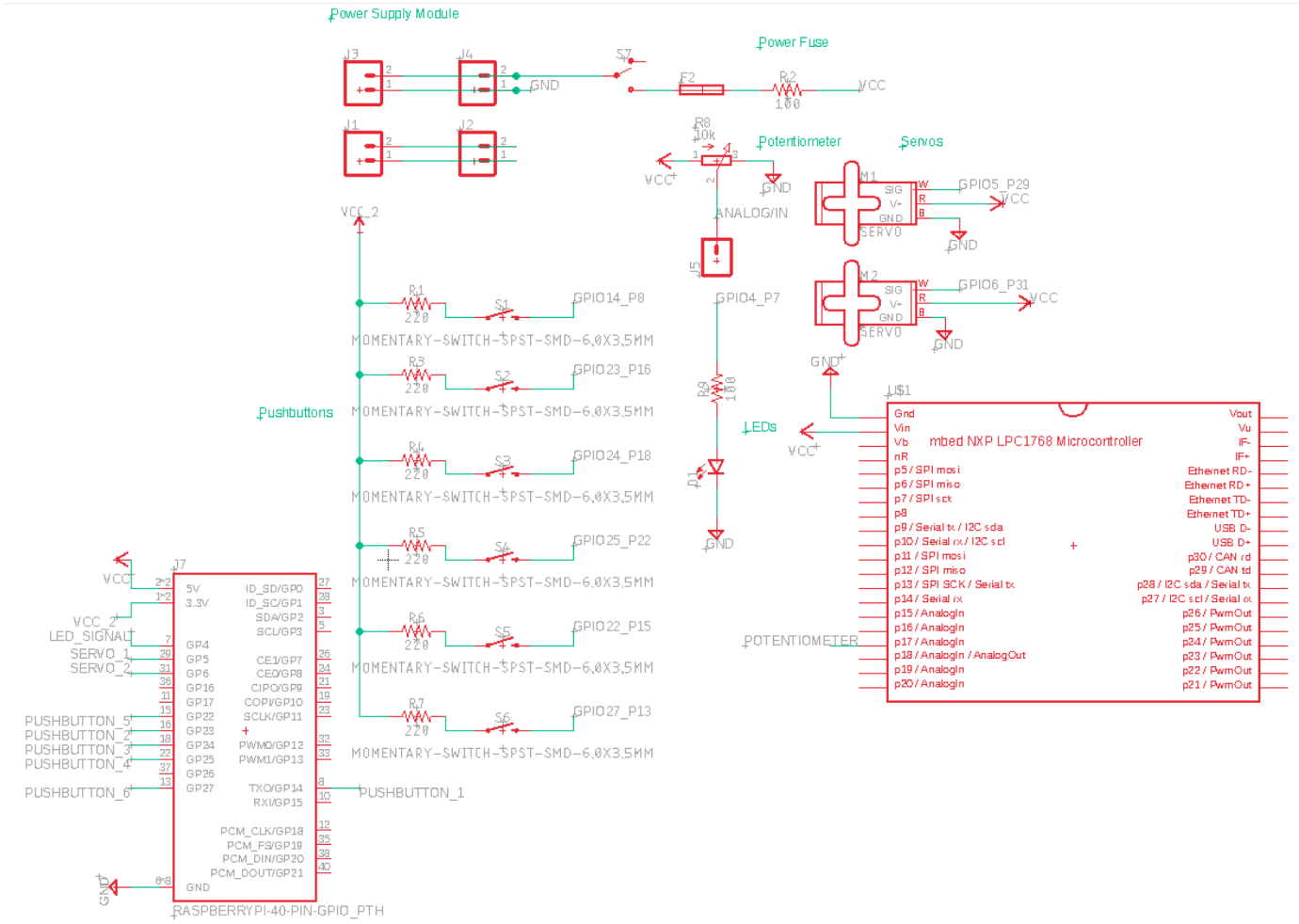

Figure 18: The actual layout of our design created in Eagle with Raspberry PI and Mbed pin assignments and components that we will make after receiving the PCB. We will be using heat shrinks and housing for the wires. The solder of the components will also take place manually.

Using the schematic, we also created the PCB design from Eagle. This is a two-layer PCB design with our team name, class section, and product name on the silkscreen.

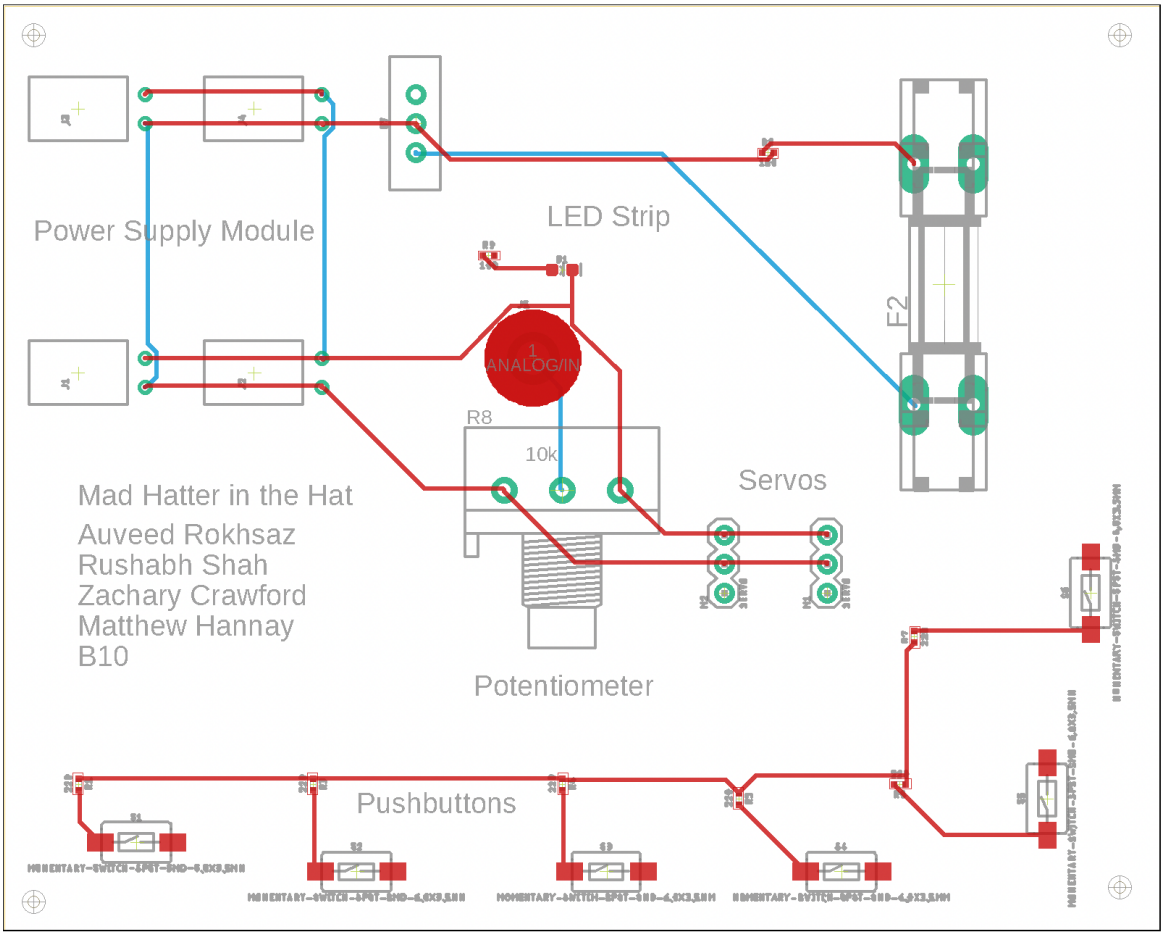

Figure 19: PCB created in Eagle for the electronic components and connections to the processors.

Using the PCB design from Eagle software, the initial PCB was created from the Invention Studio with our team name, class section, product name, and electronic component placements laser engraved as the silkscreen without the solder mask. As shown in the following figure, using the Eagle resistors created surface mount pads for the resistors. While, checking for connections, the PCB was shorting out with the incorrect soldering placements.

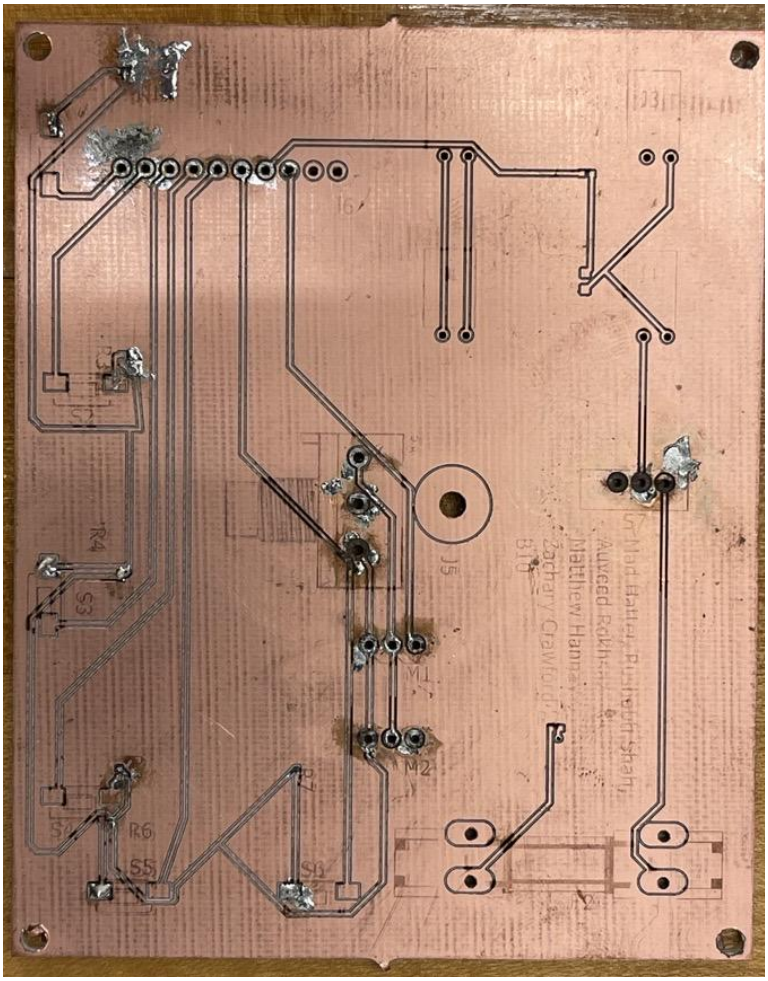

Figure 20: Initial PCB created in PCB Mill for the connections to the electronic components and processors with a silkscreen by laser engraving.

After adjusting the Eagle software for through holes, the finalized PCB was created from the Invention Studio without the silkscreen and solder mask. For the following figure, everything was correctly placed with through holes for resistors, potentiometer, servos, switch and pads for all other connections. While, checking for connections, this PCB was still shorting out with the little to no incorrect soldering placements.

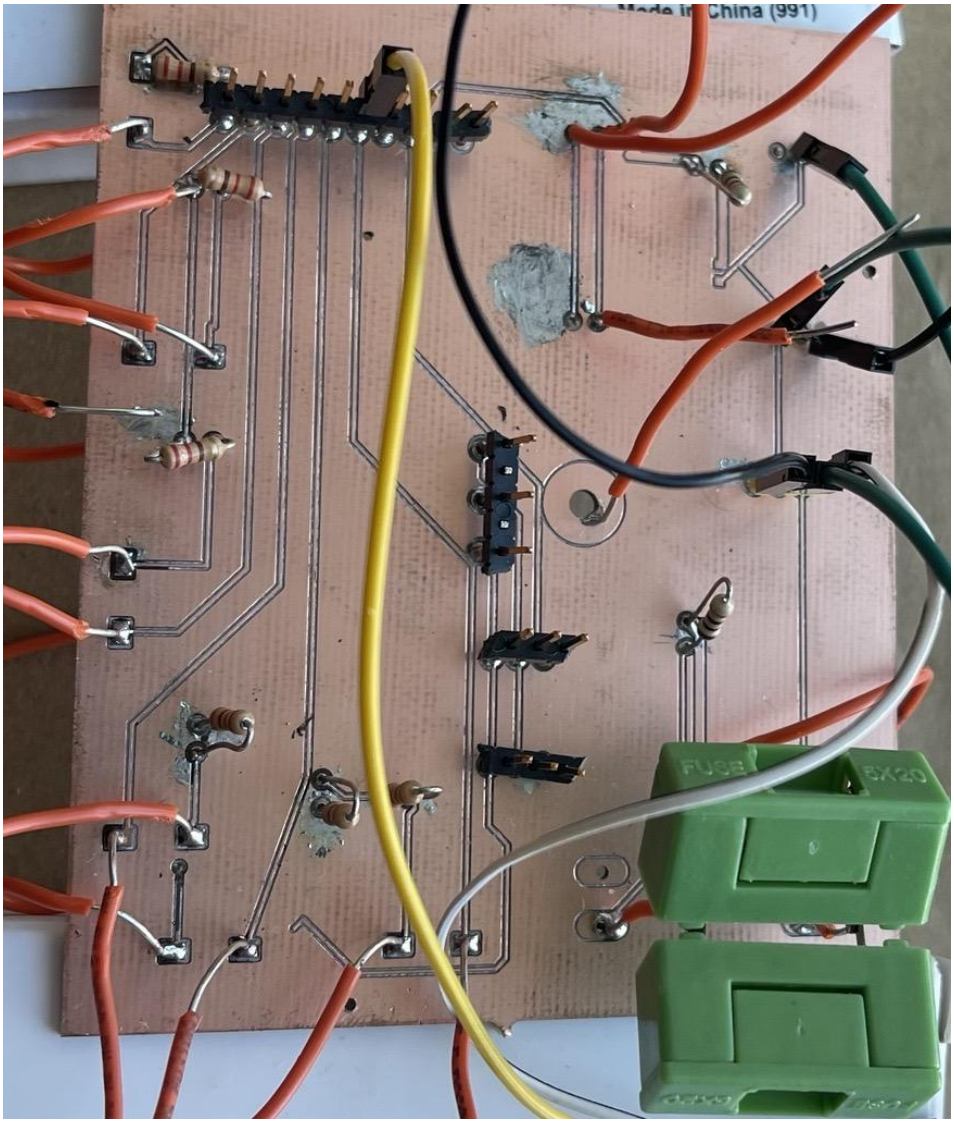

Figure 21: Recreation of the PCB from the PCB Mill for the connections to the electronic components and processors without a silkscreen.

Mechanical Design

Our mechanical design features two moving parts: the rotating teacups and the raising hat. The teacups are on a ring that is level with the raised platform and are rotated by a servo. A piston-like mechanism will push the hat to reveal the Hatter standing behind.

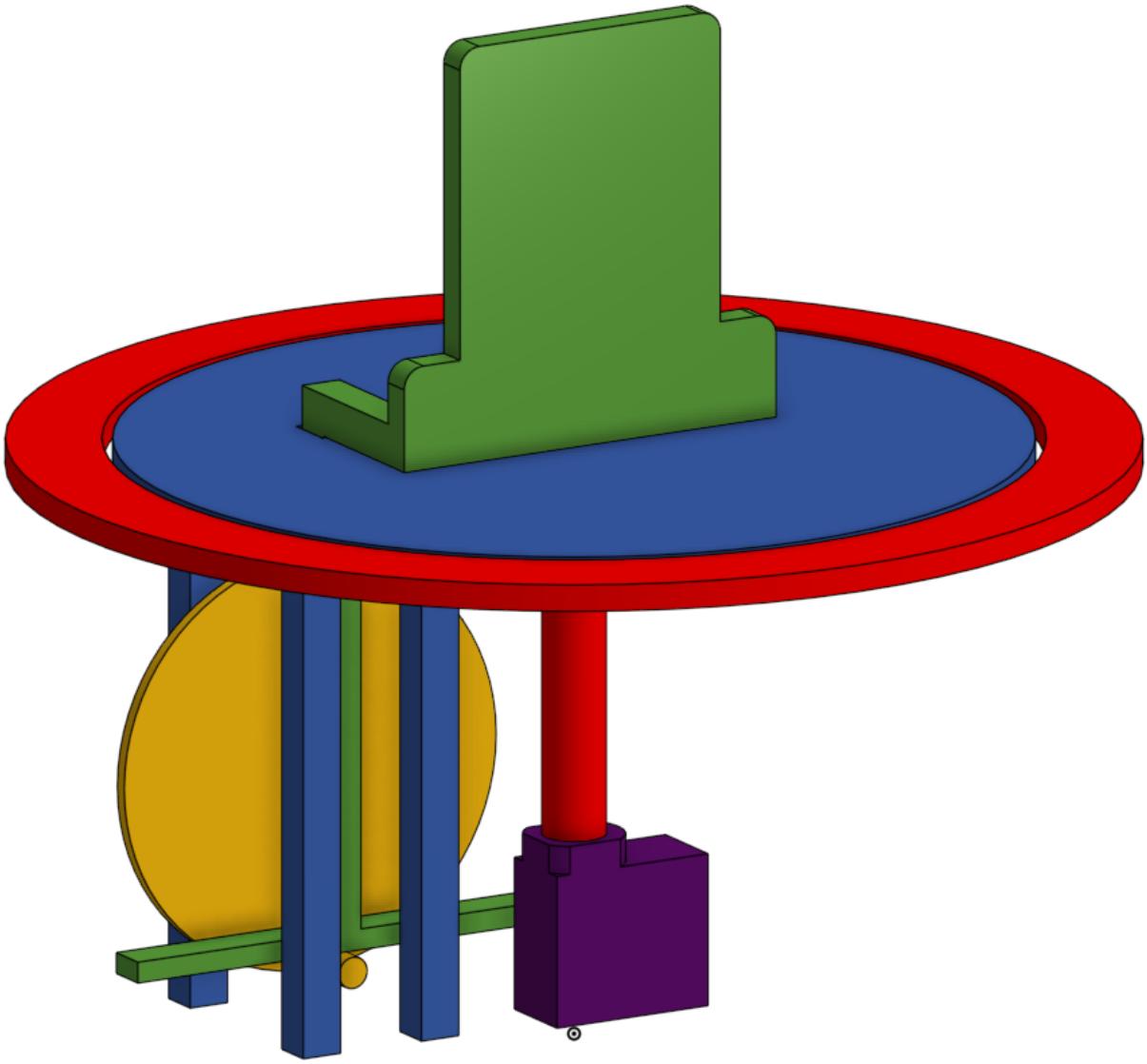

Figure 22: Front view of the main mechanism CAD

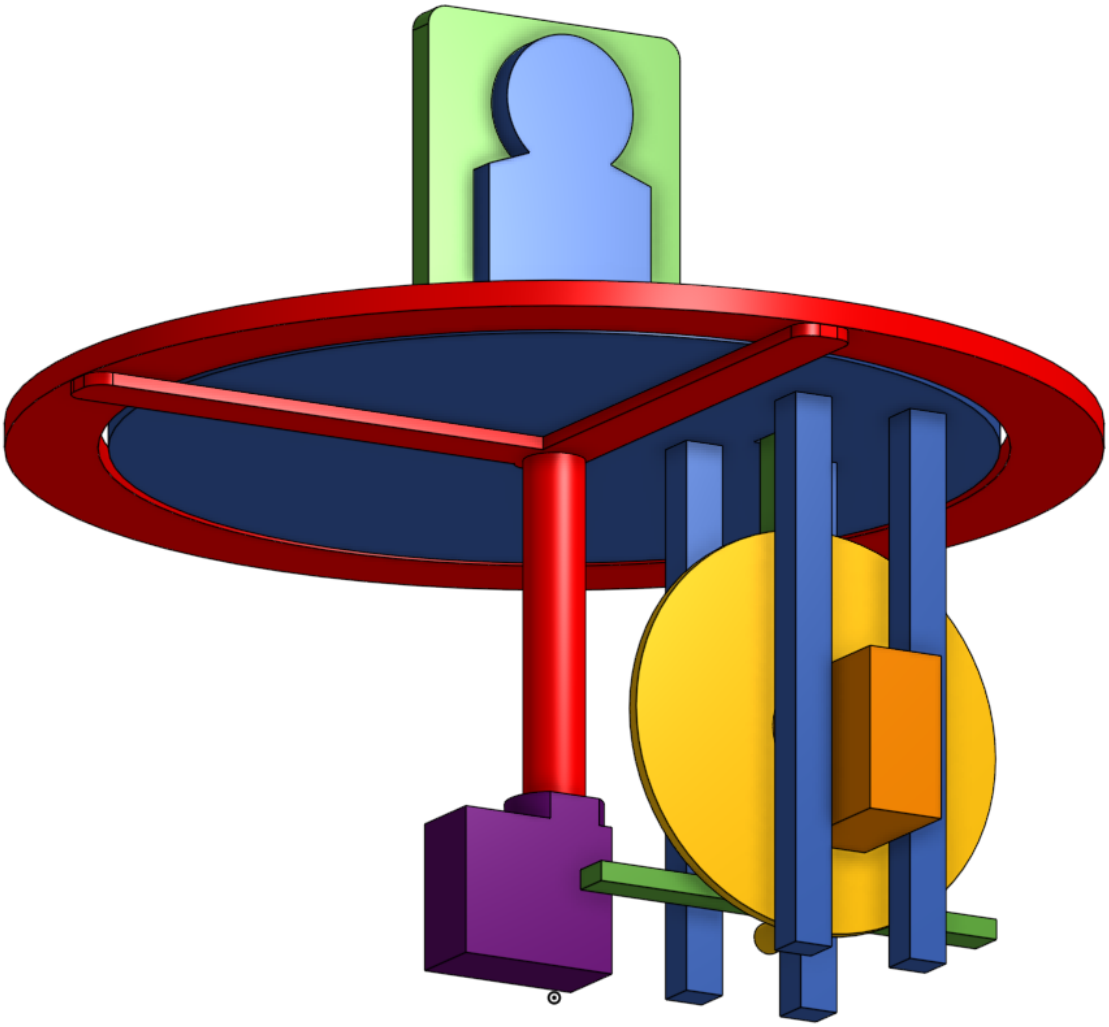

Figure 23: Back view of the main mechanism CAD

The central part of the mechanism will be held up by four support rods. This is also where the Hatter will be standing. The hat will be lifted by the wheel rotating 180 degrees and pushing the rod upwards with a peg. The ring will be driven by a servo to rotate 90 degrees at a time to present 3 different teas to the viewer. For this many moving parts, weight is a concern. The ring, hat lifting rod, and upper part of the platform will be made of light wood. The Hat and the Hatter will be laminated paper.

The front of the thespian will have a speaker for audio and six different buttons. The buttons will be used for testing purposes, and activate the following:

- Test the rotating tea cups
- Test the rising hat
- Test the lights
- Test the audio
- Run the complete sequence
- Reset

Mechanical Rotation

To control the rotating teacups, a servo (the purple block in Figures 22 and 23) will rotate a ring system (red). This servo will rotate in 90-degree increments with some time between rotations. When reset it will return to its starting position.

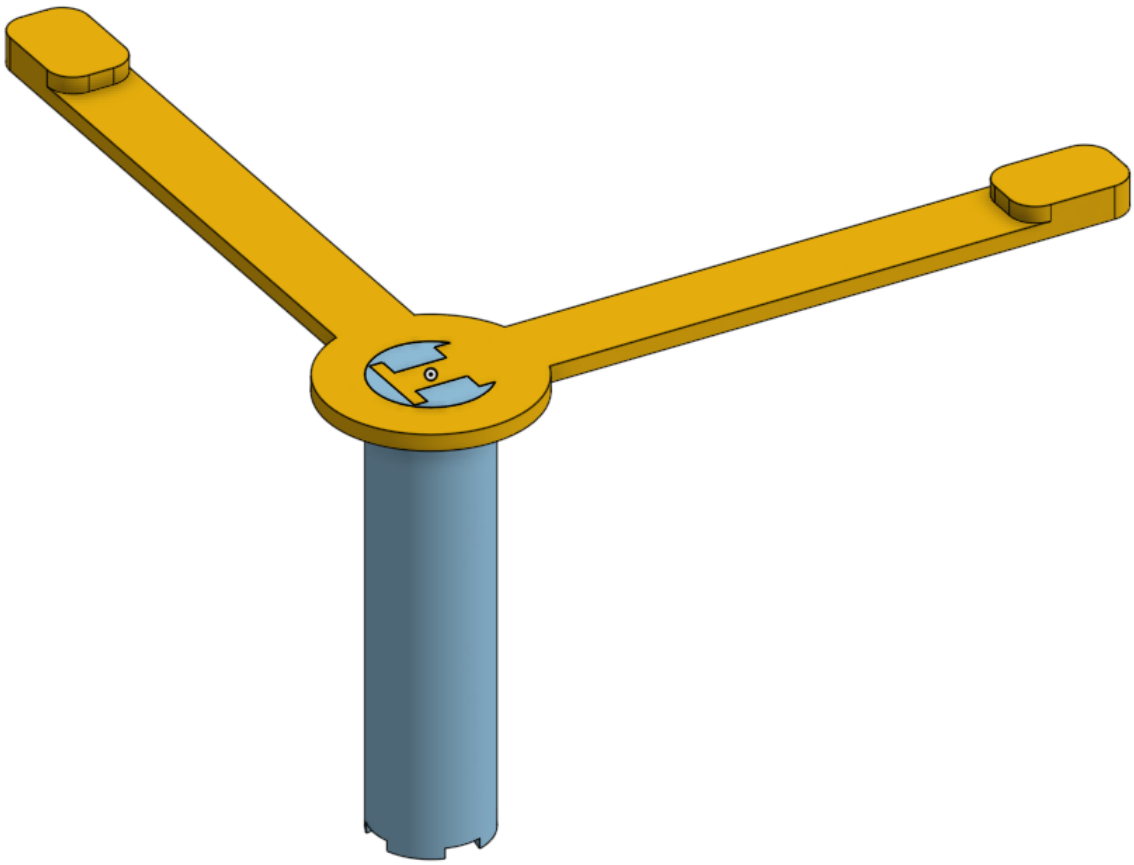

Figure 24: The connections between the servo and the ring. Note the holes at the bottom of the vertical rod to fit the servo horn.

Mechanical Lifting of the Hat

A servo (orange in Figures 22 and 23) will be used to lift the hat upwards. The wheel with a peg (yellow) will lift the hat (green) when it gets rotated 180 degrees. On reset, the servo will return the hat to the lower position.

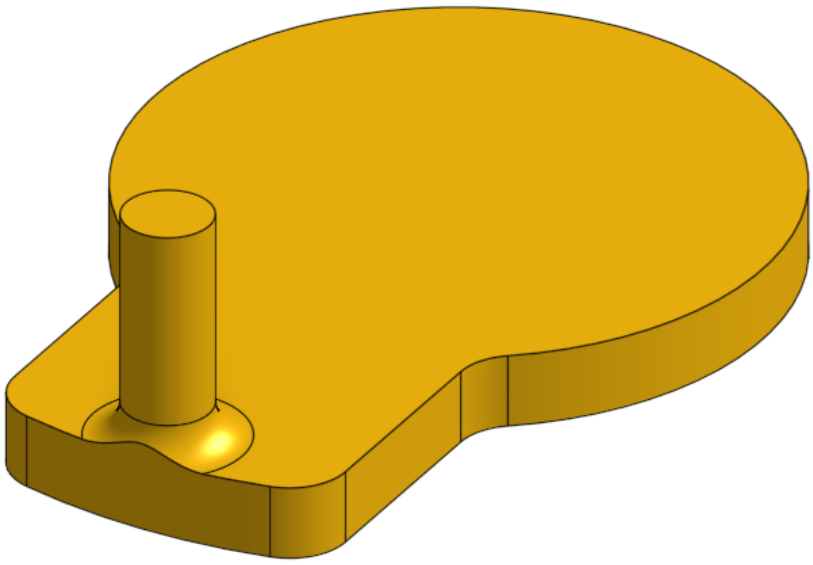

Figure 25: The wheel component of the lift subsystem. There is a servo-horn shaped hole the design to allow for it to connect to the servo.

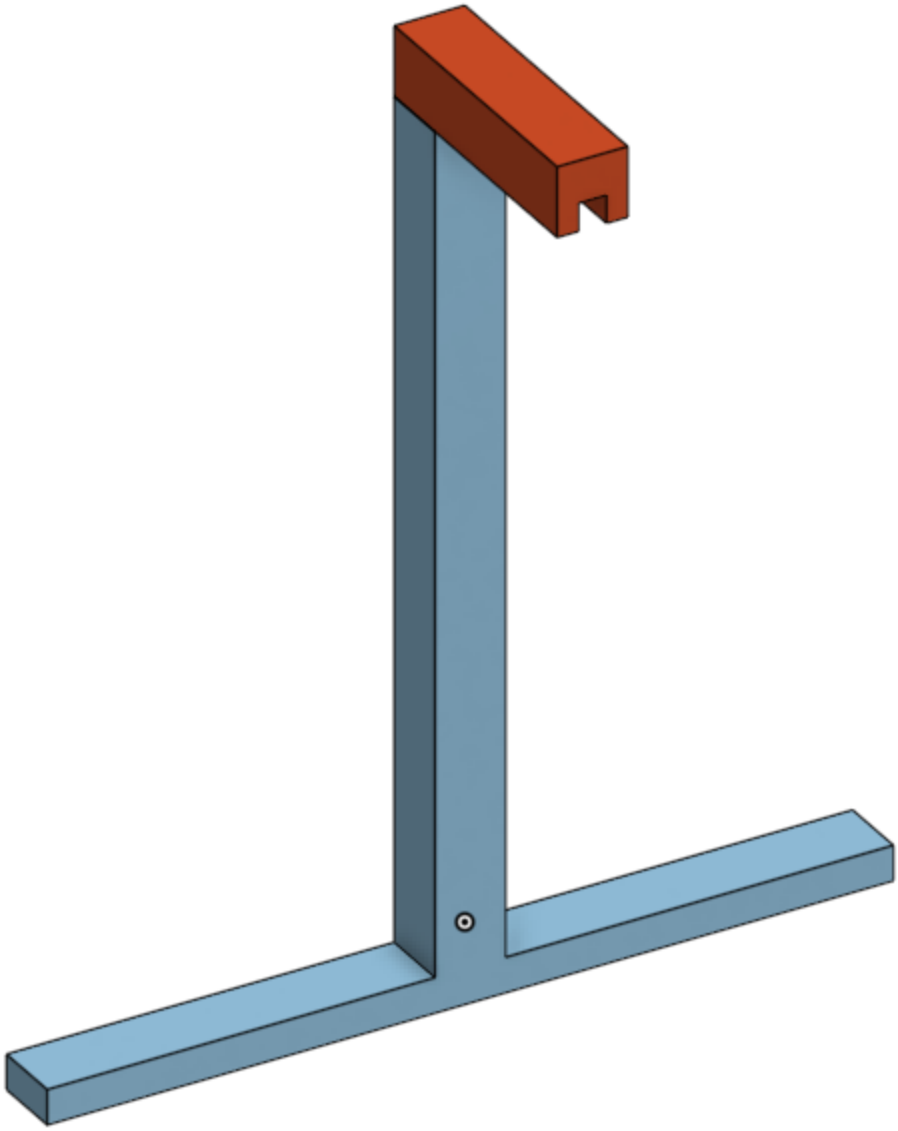

Figure 26: The lifting bar component of the lift subsystem. The width of the bottom means that it will be lifted even when the wheel pin is all the way to the side. The component is split into two so it can be fit into the hole at the top of the table and then combined.

Mechanical Testing and Integration

For simulation testing, we will verify that the servos are able to move their loads before integrating them fully into the system. The servo that rotates the teacups will be placed vertically with the rod and ring attached. The test will pass if the servo is able to rotate in 90-degree increments from 0 to 180 degrees. The servo that lifts the hat will have the wheel and rod system set up with a human tester holding the rod system in alignment. The test will pass if the servo is able to rotate 180 degrees and lift the rod to its max height.

Once these unit tests are complete, the servos will be ready to be placed into their housings. The thespian will be designed to allow for easy access to internal parts. The top and side of the table and the top of the box are parts that will be separable from the rest of the system.

Software Design

Software State Machine System

Our software design includes 9 different states—**Idle**, **Connect To Director**, **Wait For Instruction**, **LED Test**, **Audio Test**, **Motion Hat Test**, and **Motion Teacups Test** states recorded shown in Figure 27. The design centralizes around an **Idle** state that waits for user inputs. If the *Play All Button* is pressed, the system waits for instructions from the Director to play its respective lines and perform designated movements. To access the test states, each test mode has a button that can be pressed to test the respective subsystem. At any point, the *Reset* button will generate an interrupt on the processor that will return the system to the **Idle** state. Figure 28 describes the software design hierarchy.

Software Hierarchy

A **current_state** object handles the movement from one state to another. Each subsystem has its own thread that is responsible for running the methods associated with each subsystem for that state. For example, in the *PERFORM_ALL* state, all of the threads (the *audio_thread*, *motion_hat*, and the *motion_cups* threads) run their designated functions. For the test states, the functionality of each subsystem is isolated using the threads, and only the designated subsystem is tested. For example, pressing the audio test button on the interface will only run the *audio_thread* state, which controls the audio subsystem. Figure 28 shows how each state interfaces with each helper method and subsystem thread hierarchically. Threads and higher up controlling methods are shown in gray bubbles while methods controlling the device on the component level are shown in blue. Notice that controller methods may need to access different lower level methods to allow for the functionality of a specific state. The *reset_callback* function is an interrupt function that will interrupt whatever routine is occurring and return the system to the idle state.

Software Simulations and Integration

A series of software simulations were designed to verify the software of the device is functional during the development of the system with respect to the hierarchy shown in Figure 27. A demo program was created for each simulation. Note that because the Power Subsystem is solely hardware based, there are no simulations for it in software. **Please note that all the mentioned files are available on the Github link for the project:** https://github.com/auveedgatech/madhatter.

Audio Sub-System

- Speaker Audio Simulation
    - The *speaker.py* file tests the audio output to the USB speaker using a python library called *pygame.* This package converts the audio in the form of a *.wav* file

to an USB signal that is sent to the Raspberry Pi's USB port. The speaker, which has an USB input, then plays the audio sent.
    - We found that the speaker quality is better when a high quality speaker is connected via an USB cable as opposed to an analog speaker driver system. Also, the library used for the speaker, *pygame*, allows for a larger degree of control over the audio output as opposed to an analog driver because it integrates loading files with outputting them to the speaker.
- Speaker Volume Simulation
    - The *analog_speaker_volume.cpp* file tests the volume control feature of the Audio Sub-System. A potentiometer circuit is connected to Pin 20 of the mBed (AnalogIn). The mBed's built in ADC converts this to a float between zero and on based on the input voltage from the potentiometer. This float represents the fraction at which the speaker volume will play. When it is time to integrate the audio sub-system with the computing sub-system, a serial USB connection between the Raspberry Pi and mBed will be used.
    - We found the accuracy of this percentage with respect to the degree in which the potentiometer was turned is high.
- State Machine Integration Test
    - The *software_demo_code.py* integrates the use of the speaker with the overall state machine. When the correct pushbutton is pressed, the speaker plays ten seconds of audio.
    - We found that the speaker library *pygame* allows for a large degree of software control in the form of play, pause, and stop functions. It makes it easy to load different audio files and interchange them quickly.

Interface Subsystem

- Pushbutton Simulation
    - The *pb.py* file tests the use of a pushbutton through the Pi's GPIO system. This file simply prints an output on the press of a button.
    - This test solidified the use of buttons in our final design as opposed to switches.
- Inputs in the State Machine Simulation
    - The *software_demo_code.py* integrates the use of push buttons as inputs to the state machine. Interrupt functions are used to change variables in the software system in real-time. See the Computing Sub-System for more details.
    - We found that emphasis on button debouncing is unnecessary if using a state machine approach because the callback functions only change state, a state cannot be re-entered upon entry.

Motion Sub-System

- Servo Simulation
    - The *Servo.py* file tests the motion of the servos used for moving the mad hatter's hat as well as rotating the teacups. This file was used to test the servos with the actual mechanical components they are attached to in Figure 23.
    - We found that the servos are jittery, but developed a manual method of debouncing using the Pi's GPIO.
- Movements in State Machine Simulation

- The *software_demo_code.py* integrates the use of servos into the state machine. This file ensures that the servos can rotate to their maximum as well as reduce jitter when the reset button is pressed. Furthermore, this file tests the accuracy of the angular rotation of the servo by increments of 90 degrees.
- In this file, we created a function that is operable on any PWM based servo. For our final design, we will build a software API that, regardless of the servo, will be able to perform a series of functions correlating with rotation. For example, we will write a rotate ninety degree function, which will rotate a servo clockwise or counterclockwise based on the input.

Light Sub-System

- LED Control Output
  - The *software_demo_code.py* tests that the Pi GPIO output controlling the LED strip operates correctly. When the pushbutton testing the LED system is pressed, the LED output GPIO bit goes high for ten seconds. In our simulation videos, a singular LED is used to show this output bit is high.
  - This test helped us determine that using an GPIO bit is the best way to control the LED bit. From software, we can integrate the LED into virtually any function.

Computing Sub-System

- Software State Machine
  - Each of the previous subsystems have detailed how they are integrated with the computing subsystem in the *software_demo_code.py* file. This file has a built-in state machine which acts as the controller for the entire hatter system. The different states are detailed in Figure 27. Idle is the home state from which all of the other states are reachable from. Pressing the reset button will always take the user back to the Idle state using a Python function interrupt. From the Idle state, the other test states described above are available through the use of push buttons from the interface subsystem.
  - We found that the software state machine is by far the most stable way of integrating the subsystems together. Trying to use a file system with different scripts for each subsystem is not only slower (because of loading times), but also prone to more edge case errors such as debouncing. The state machine approach allows for easy management of the user's inputs.
- Director Code Simulation
  - The director code (given by instructors) is used to connect to the Mad Hatter through a local network. We were able to register the robot with the director.
  - We found that if the director and robot scripts are both running on the Pi, the director is able to connect and send commands to the robot script. However, we found that if the director was a different device, after registering with the director, the robot is unable to receive commands and actually crashes. This was unexpected, and we will be working to solve this issue shortly.

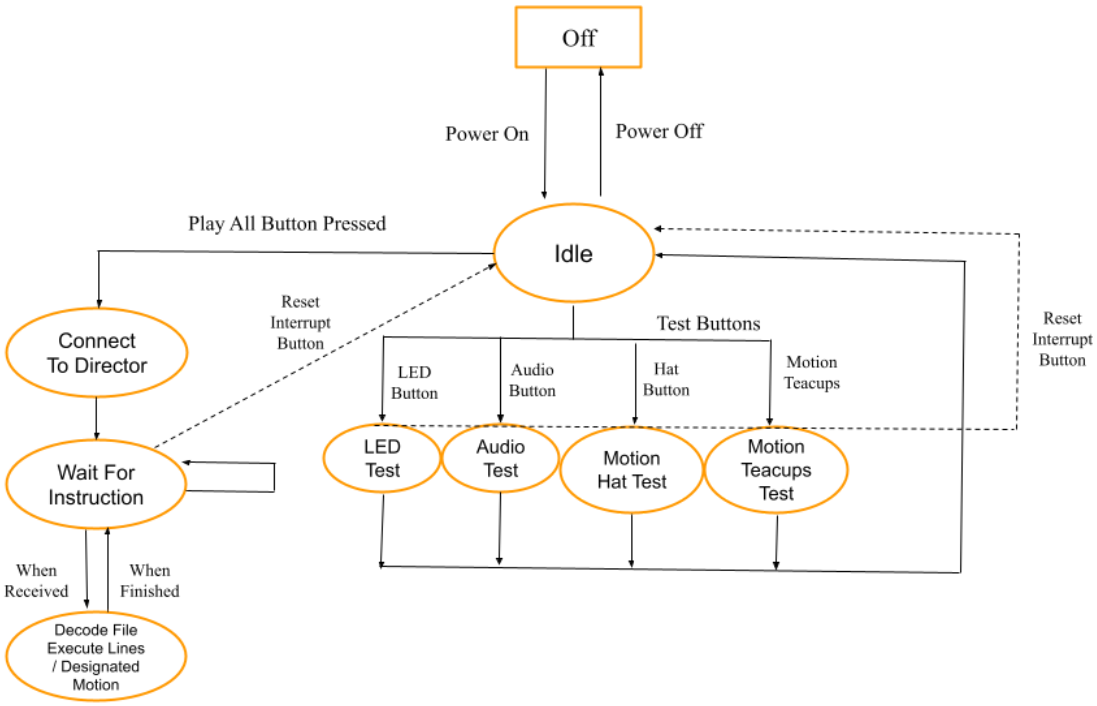

Figure 27: Software state machine and flow with test states, instruction states, and an idle state.

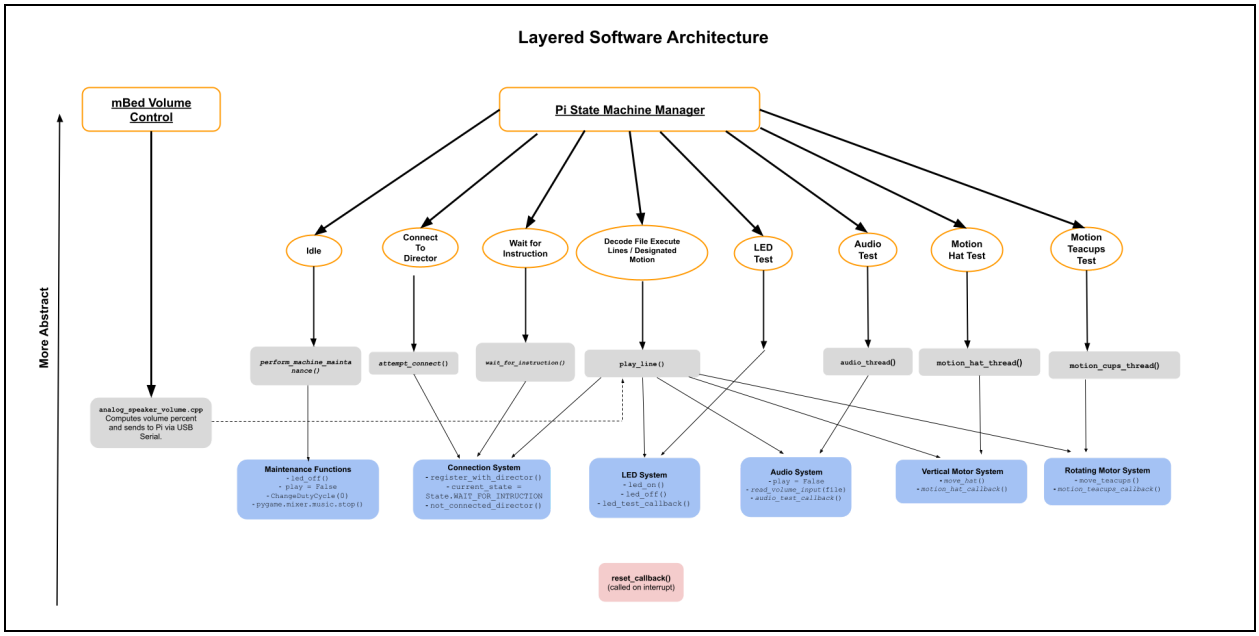

*Figure 28: Layered Software Architecture. The **State Machine Manager** manages the functionality of the system while the **Debug Manager** manages to access debug data to help developers troubleshoot issues that may occur. Each of the different states accesses different system methods to achieve their function. The reset button connects to the on_reset button to interrupt any routine and return to the **Idle** state at any time.*

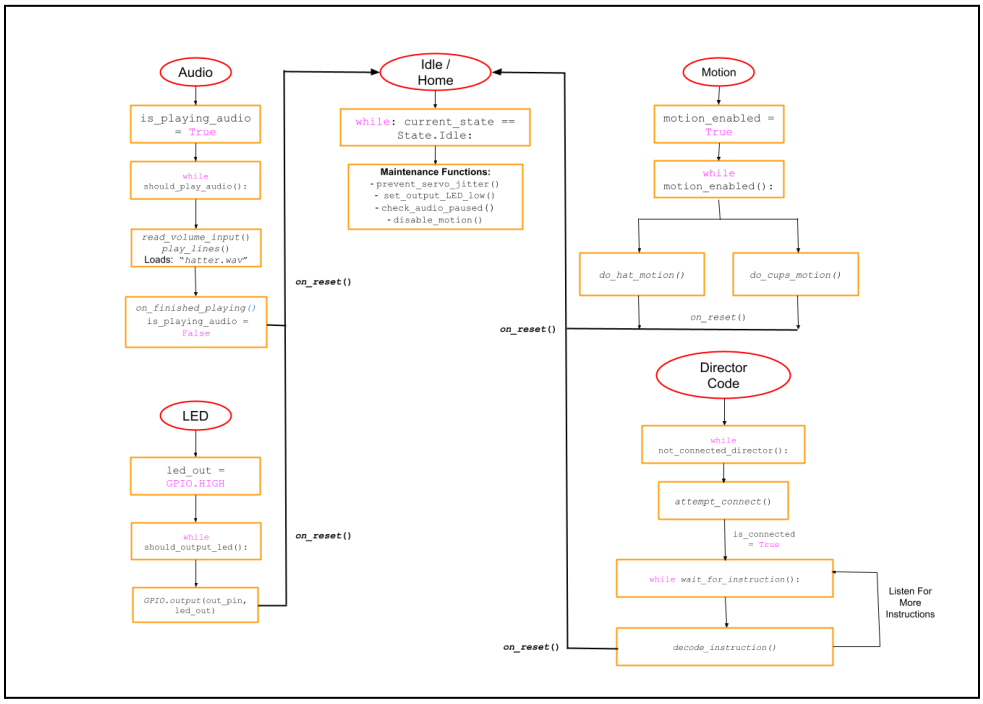

Figure 29: Details the functionalities associated with each subsystem. This diagram shows how each subsystem functionality interfaces with the software state machine.

Schedule / Subsystem Leaders

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

Task	Week 6: 09/25	Week 7: 10/02	Week 8: 10/09	Week 9: 10/16	Week 10: 10/23	Week 11: 10/30
SOFTWARE SUBSYSTEM						
Software Design Director	Auveed					
Software Design User Inputs	Auveed					
Software Design Motion and Lights	Auveed					
Software Design Audio	Zachary					
Software Build User Inputs		Auveed				
Software Build Motion and Lights		Matthew				
Software Build Audio		Zachary				
Software Test User Inputs			Auveed			
Software Test Motion and Lights			Matthew			
Software Test Audio			Zachary			
ELECTRICAL SUBSYSTEM						
Hardware Design User Inputs	Rushabh					
Hardware Design Motion and Lights	Rushabh					
Hardware Design Audio	Zachary					
Hardware Design PCB		Rushabh				
Hardware Design Power		Rushabh				
Hardware Build User Inputs		Zachary				
Hardware Build Motion and Lights		Rushabh				

Hardware Build Audio		Zachary				
Hardware Test User Inputs			Zachary			
Hardware Test Motion and Lights			Rushabh			
Hardware Test Audio			Zachary			
MECHANICAL SUBSYSTEM						
Mechanical CAD Design User Inputs	Matthew					
Mechanical CAD Design Lights		Matthew				
Mechanical CAD Design Structure		Matthew				
Mechanical CAD Design Mbed Holder		Matthew				
Mechanical CAD Design Servo Motors		Matthew				
Mechanical CAD Design Switch		Matthew				
Mechanical CAD Design Potentiometer		Matthew				
Mechanical CAD Design Power		Matthew				
Mechanical Build User Inputs			Matthew			
Mechanical Build Motion, Lights, Audio			Matthew			
Mechanical Test User Inputs				Matthew		
Mechanical Test Motion and Lights				Matthew		
Mechanical Test Audio				Auveed		
SYSTEM						
PCB Soldering and Fabrication				Rushabh		

System Integration Motion, Lights, Audio					Rushabh	
System Integration Mad Hatter Structure					Rushabh	
System Test PCB						Rushabh
System Test Motion, Lights, Audio						Rushabh
System Test Mad Hatter Structure						Rushabh
System Test Power						Rushabh

Table 2: Schedule to complete Mad Hatter (One-Month timeframe). Weeks are relative to the master schedule.

Integration

Each subsystem will need to be able to work together with others. The subsystems will need to not only perform their designated tasks but also be able to work in conjunction with one another. For each subsystem, software and hardware have been tested together on a breadboard. The subsystem with a mechanical load has also been tested. After the subsystems have been fully tested, the code for that system will be refactored into an API for that subsystem and the electrical circuit will be incorporated into a PCB. This preliminary test will be isolated, meaning it will not incorporate the other subsystems in the design. After all of the preliminary tests are done on each subsystem, the subsystems will be tested in conjunction with a power supply under similar conditions to the actual device (wall plug 120V AC). After a PCB is designed for each subsystem, the PCB will be tested in a simulated environment (ideal power settings) and then in an environment with all the other subsystem PCBs as part of a larger integration test. Mechanical loads are incorporated into this test.

Software High-Risk Parts:

- A python state machine must ensure that timing parameters for each electrical component is met. Timing is important for resetting the device, responding to network messages, and managing the subsystems in real-time is important. Each test and simulation will ensure timing parameters are met.
- Power to the microprocessor is paramount because the microprocessor handles all the different states of the device. If power is lost the system fails.
- The reset button must instantly exit any state. This requires the use of an interrupt function to interrupt the current task. If the system does not reset correctly, the entire state machine can become stuck in a loop.

Hardware High-Risk Parts:

- Servo motors are another high-risk part of the design. They require the most current and are the most visually interesting aspect. If the servo motors fail to run, then Mad Hatter's hat will not be lifted up to his headline and the overall design will not run as intended.
- We still have the PCB components to test and revise our design with. Knowing we only have these tests available, we want to reduce the risk of failure when soldering our parts together. To reduce risk, we have been to the lab several times to breadboard the speaker, push buttons, motors, LEDs, and the USB Mini Stereo Speaker. By debugging our wiring and getting a sense of how each part connects, we will be more confident when going into the PCB and soldering stage.

Mechanical High-Risk Parts:

- There are two motors, one for rotating the teacups and one for moving the hat vertically. Each motor must be able to handle its respective load and each mechanical subsystem will be tested to ensure that it can perform its delegated tasks.
- The overall structure of the design (the box around it) must meet design requirements as well as fit all of the PCB boards inside.

Repository Management:

- We have a GitHub that includes our electronic system files (schematic, PCB, and Eagle files). The GitHub repository also includes our software code.
    - https://github.com/auveedgatech/madhatter
- We also share design documents and files in our central Microsoft Teams SharePoint folder

Configuration Controls:

- The UI includes a series of push buttons as well as a power switch. This will be integrated into the design using the RTOS.

Conclusion / Current Progress

We have completed our Mad hatter project. By breaking the project into subsystems and creating smaller and more feasible tasks, we are able to distribute and complete these tasks in a feasible manner. We have tested and simulated all the subsystems on the breadboard and have designed and soldered the PCB layout. The subsystem simulations has been integrated into the larger state machine subsystem as well.

Over the course of 12 weeks, our team was able to design the Mad Hatter Box. We divided our team into three parts: a software team, a hardware team, and a mechanical team. Our team faced many challenges in producing the Mad Hatter. In the electrical subsystem, we were not able to produce a functioning PCB. Rather our design required a breadboard setup to connect the Pi to the different subsystems. Furthermore, we decided to include an mBed instead of an analog to digital converter to process the volume control input, which added to the total cost of the project. Furthermore, because our PCB was not functional, we were not able to perform many of the electrical tests found in the test plan. Furthermore, we lacked a unified power system and regulation due to time constraints. Mechanically, we relied heavily on non-glued 3D printed

and laser cut parts. This meant that in the final projects, there were some components that did not fit together and had to be glued or taped. In the software, we encountered issues with the sampling of serial data from the mBed. The serial connection would often lag leading to a delay in the change of volume control.

If given this project again, there are a few things that we would have done differently. We would start by designing the system in CAD. We would then breadboard each subsystem and then integrate them into one larger system. Then we would integrate the circuits with the mechanical features. Rather, we used a more scattered approach, developing software for the overall system.