# Design Document

*The Rapid Rabbit*

*Team #C01: Michael Landon, Bailey Rende, Emma Rae Stanley, Rhea Prem*

**Date: December 6, 2022**

## Table of Contents

Project Description ....................4
System Design ....................4
Sub-System Designs ....................6
  Software Controller Sub-System ....................6
  Audio Sub-System ....................7
  Mechanical Sub-System ....................7
  Power Sub-System ....................7
  Lights Sub-System ....................8
Constraints, Alternatives, and Trade-Offs ....................8
Electronic Design ....................9
Electronic Simulation ....................12
Mechanical Design ....................12
Mechanical Simulation ....................20
Software Design ....................22
Software Simulation ....................23
Schedule ....................25
Integration ....................27
Conclusion ....................28

## Revision Record

| **Date** | **Author** | **Comments** |
|---|---|---|
| Sep, 15, 2022 | Team | Document Created (system design and hierarchical design) |
| | Rhea Prem | Initial draft of Project Description, System Design, Sub-system Design written |
| | Michael Landon | Revised System Diagram |
| Sep. 18, 2022 | Team | Sub-system Input and Output Table made |
| | Bailey Rende | Revised System Diagram, wrote sections for audio, mechanical, and power sub-systems. |
| | Rhea Prem | Software Controller Sub-system written, and Diagram created |
| Sep. 19, 2022 | Rhea Prem | Software design and schedule portion written |
| Sep. 22, 2022 | Bailey Rende | Updated lights and power sub-system diagrams and section |
| | Rhea Prem | Updated software controller diagram |
| Sep. 24, 2022 | Rhea Prem | Updated software state diagram & description, hierarchical design diagram, system design, integration, constraint/trade-off, and conclusion sections written. |
| | Bailey Rende | Updated Electrical and Mechanical Design |
| Sep. 25, 2022 | Rhea Prem | Software Architecture Diagram and Text |
| Oct. 18, 2022 | Rhea Prem | Software Simulation section added, updated Software Controller block diagram |
| Oct. 22, 2022 | Michael Landon | Update system diagram |
| Oct. 23, 2022 | Michael Landon | Electronic Simulation section added. Electronic Design picture updated. GPIO and Power PCB sections and figures added. Integration risks |
| | Bailey Rende | Update sub-system inputs and outputs table, add robot box design and laser cut prototypes to mechanical design, update audio and lights sub-systems, update mechanical design section |
| | Rhea Prem | Update software simulation section |
| | Emma Rae | Add mechanical simulation |
| Dec. 4, 2022 | Rhea Prem | Add conclusion |
| | Bailey Rende | Add final box design, rabbit attachment design, and final ear design to mechanical section |
| Dec. 6, 2022 | Bailey Rende | Update Table of Contents, proofread entire document |

## Project Description

Our team has designed an Alice and Wonderland inspired White Rabbit robot. In our design, the robot will receive lines from a “Director” (in this case, an established server), and, in response, the robot will start playing sounds through a speaker, 12 LED’s will illuminate in a clock face, and the rabbit’s ears will rise and lower.

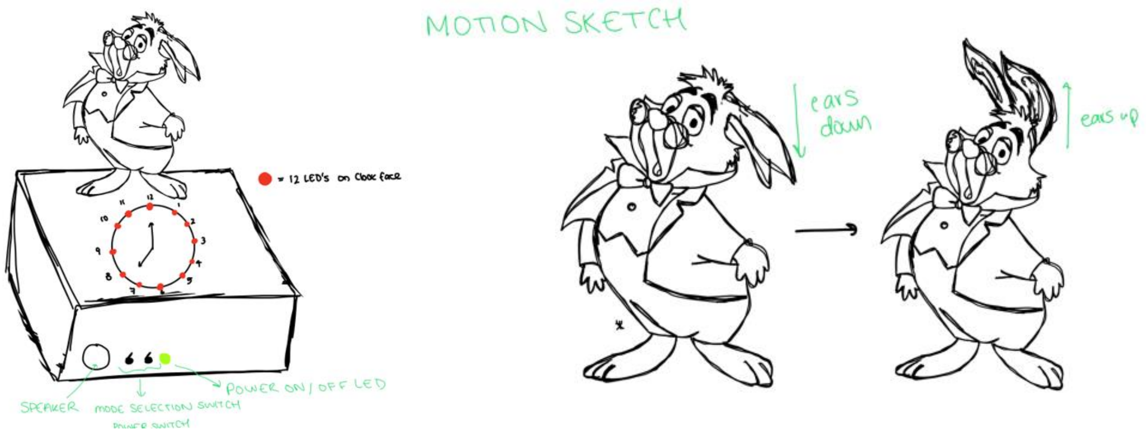


*Figure 1: Initial Sketches of the Rapid Rabbit*

*Table 1: Task Leads for Each Sub-System*

| **Task Leads** | |
|---|---|
| **Sub-system** | **Lead** |
| Mechanical | Emma Rae and Bailey |
| Software Controller | Rhea |
| Lights | Michael and Bailey |
| Power | Michael |
| Audio | Bailey (Team Leader) |

## System Design

The system design for the Rapid Rabbit includes 5 different sub-systems. The inputs of the system include 120V AC input power, an analog audio source, and user input.
The outputs of this system include a USB speaker and the movement produced by the figurine driven by servo motors. The system consists of five sub-systems including the software controller, mechanical, power, lights, and audio. The software controls will consist of a Raspberry Pi 4 which controls the mechanical, lights, and audio sub-systems. The mechanical sub-system includes motors to control the motion of our robot. The audio sub-system includes a 3W USB speaker with an auxiliary input. The power sub-system includes a 120VAC input and output of 5V DC. It consists of an AC-DC power supply, DC-DC buck converter, user input switch, and fuse.

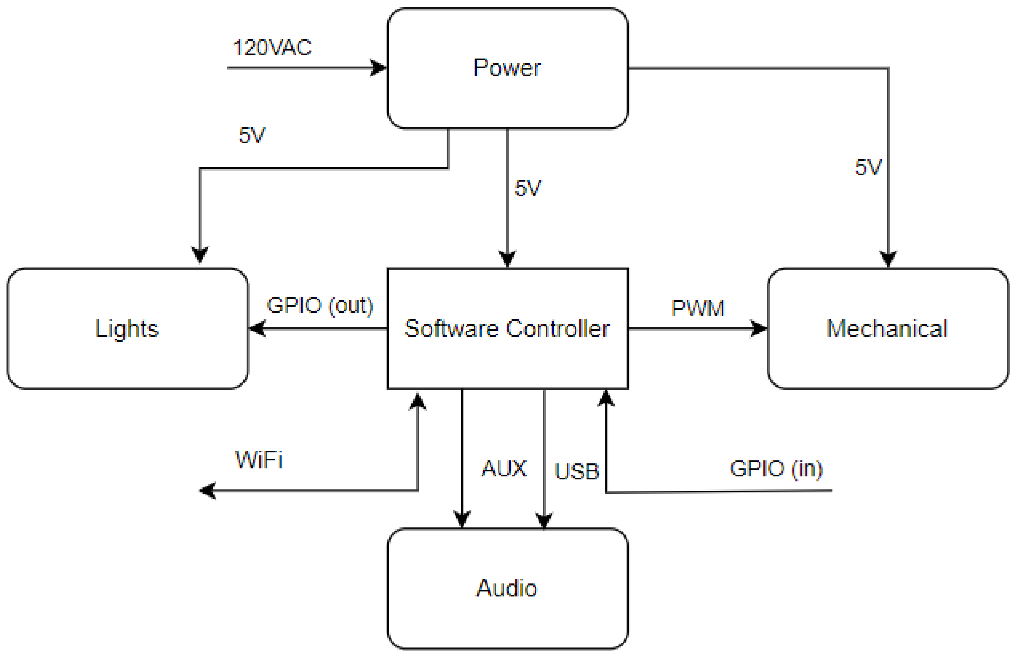


*Figure 2: System Diagram*

*Table 2: Sub-system Inputs and Outputs*

| **Interface** | **Source** | **Destination** | **Description** |
|---|---|---|---|
| 120V | System Input | Power | 120V AC power input |
| 5V | Power | Software Controller | 5V DC power |
| | | Mechanical | 5V DC power |
| | | Lights | 5V DC power |
| USB | Software Controller | Audio | Powers 3W speakers |
| AUX | Software Controller | Audio | Outputs audio to 3W speakers |
| Wi-Fi | System Input | Software Controller | Wi-Fi internet connection |
| PWM | Software Controller | Mechanical | PWM signal to control motors |
| GPIO (out) | Software Controller | Lights | 3.3V DC power for LEDs |
| GPIO (in) | System Input | Software Controller | Input from switches and buttons |

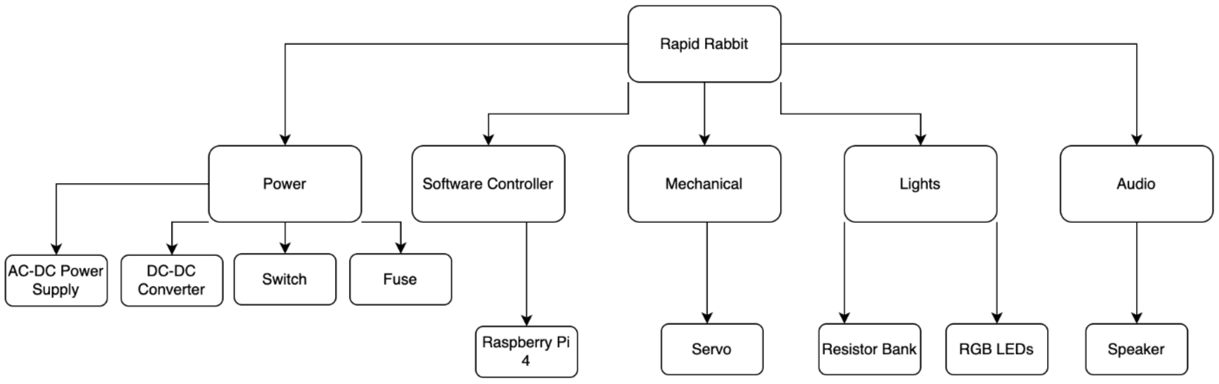


*Figure 3: Hierarchical Design Diagram*

## Sub-System Designs

Our design incorporates five sub-systems: software controller, mechanical, power, lights, and audio. The software controller will integrate all sub-systems by receiving input from the “director” and issuing output signals to other sub-systems. Audio will amplify sound in response to receiving a line, motion will execute the upwards and downwards movement of the ears, and power converts 120V AC to 5V DC for input to other sub-systems.

### Software Controller Sub-System

The Software Controller sub-system will focus on the interaction between the Raspberry Pi 4 and other sub-systems. This sub-system takes in a 5V DC input and Wi-Fi inputs in order to connect to the Director and receive lines for the robot. The sub-system then outputs a PWM signal to control the motors in the mechanical sub-system, USB to output sounds to a speaker in the audio sub-system, and 3.3V DC GPIO signals to control the LEDs in the light's sub-system.

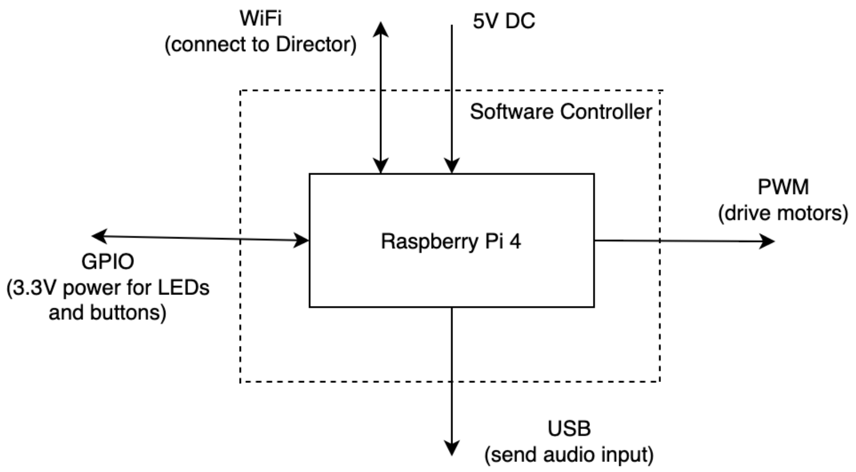


*Figure 4: Software Controller Sub-System*

## Audio Sub-System

The audio sub-system will connect via USB and AUX to the software controller for power and audio respectively. It'll then connect directly to 3 Watt speakers. This sub-system runs on 5 Volts DC.

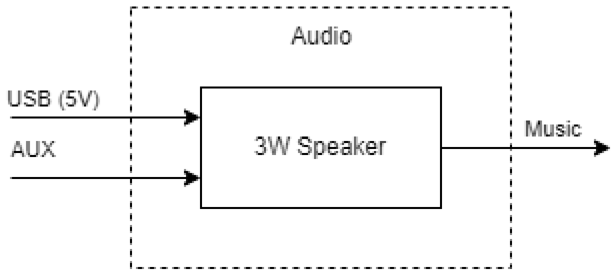


*Figure 5: Audio Sub-System*

## Mechanical Sub-System

The mechanical sub-system will take 5 Volts DC and receive two PWM signals to control the servo motors. The servo motors will then interpret the signals and power the motors appropriately to produce the desired motion.

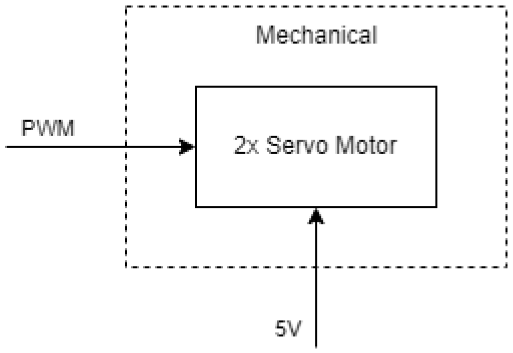


*Figure 6: Mechanical Sub-System*

## Power Sub-System

The power sub-system will take 120 Volts AC as an input and output 5 Volts DC. It will route power through an AC-DC power supply, a DC-DC buck converter to drop from 9V to 5V, a user input switch, and a fuse before distributing it to the other systems.

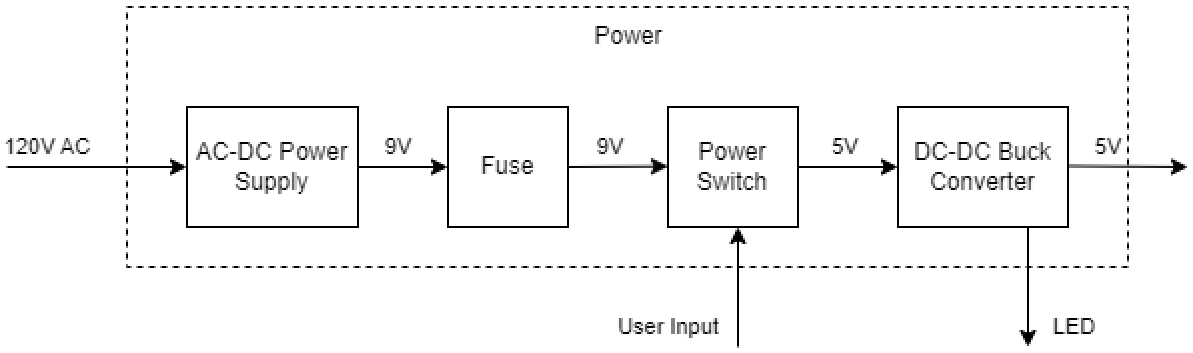


*Figure 7: Power Sub-System*

### Lights Sub-System

The lights sub-system will take 3.3 Volts DC GPIO, route each signal through a resistor, and connect it to a red component of the RGB LEDs. The lights sub-system will also take 5V DC, connect it to the collector of an NPN BJT, route it through a resistor, and connect it the blue and green LED components of the RGB LEDs. 3.3 Volts DC GPIO is also connected to the base of the NPN BJTs. All of the LEDs will then emit light.

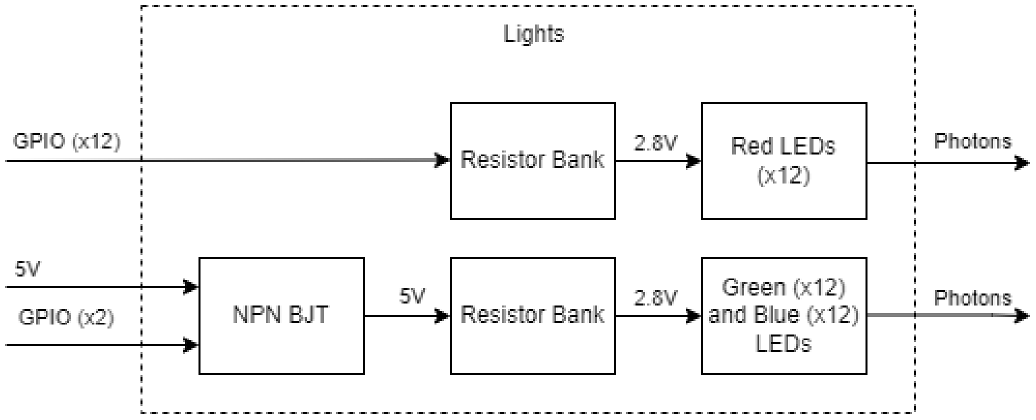


*Figure 8: Lights Sub-System*

## Constraints, Alternatives, and Trade-Offs

One constraint we had was choosing between microcontrollers. We initially were going to use the Raspberry Pi Zero, but, after having complications, we had to access trade-offs between the Zero and other controllers.
Microcontrollers: Raspberry Pi Zero vs. Raspberry Pi 4

- The Raspberry Pi Zero is very small and thin, so it doesn't take up a lot of space and is easily hidden. However, we had many problems with the microSD card not being read on our original Pi Zero. After days of trying to get the Pi Zero setup, we were unsuccessful.
- There are supply constraints with the Raspberry Pi Zero, so we were not able to find another to test. However, the Raspberry Pi 4 is more easily accessible, so we moved to the Pi 4.
- The Raspberry Pi 4 is a bit bulkier as it has 4x USB A ports, an ethernet port, and 2x micro HDMI ports, compared to the micro USB and mini HDMI the Pi Zero has. However, with the addition of a little more surface area for the Pi 4, we were able to eliminate additional connections in our design which converted micro-USB to USB A. Additionally, the Pi 4 has more GPIO pins, which will help us control all of our LEDs.

Another trade-off analysis we performed was with servos. Initially, we selected the TowerPro SG90 servos. After further testing, we questioned whether we should switch to the larger, more powerful Hitec HS-422 servos.
Motors: TowerPro SG90 servo vs. Hitec HS-422 servo

- The TowerPro SG90 servo has a stall torque of 1.8kg/cm, weights 9g, operates at 5V, and has 180 degrees of motion
- The Hitec HS-422 servo has a stall torque of 4.1kg/cm, weights 45.5g, operates at 5V, and has 180 degrees of motion

- While the Hitec servo has a much greater stall torque, this isn't needed for our application. Additionally, the Hitec servo has a stall current draw of 800mA vs. 700mA for the TowerPro.
- Despite the Hitec motor having a more durable gearing system, the TowerPro's mechanism will suffice for our application. Considering it's lighter, draws less current, and is smaller, it'll be the servo of our choice.

## Electronic Design

The electrical design consists primarily of supplying power to the components and a few connections to send signals to and from the Pi. The Pi can be supplied mode information through the control switches on its GPIO pulling low to ground. The Pi can also turn on and off the LEDs on the clock face by pulling high or low on the associated GPIO pins.

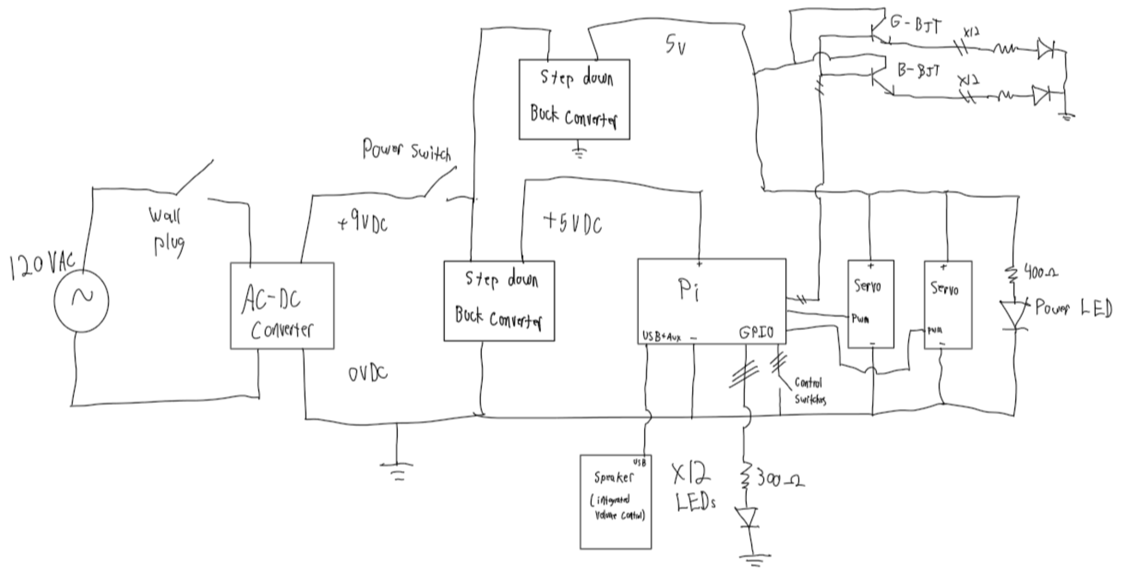


*Figure 9: Electrical Connections Sketch*

The majority of the LEDs are connected to a GPIO PCB. This PCB connects to the Pi 4 header and routs the GPIO pins to 12 RGB LEDs mounted on the board. All of the red LEDs are connected to individual GPIO pins, allowing for individual control. The green LEDs are all connected to a single NPN BJT, which is controlled by a single GPIO pin. The blue LEDs are connected the same way. This means that all 12 blue and/or green LEDs are controlled together, not individually. It also has a second header to allow tapping into the Pi pins for the servos and other elements as the ribbon cable will block access to these pins otherwise.

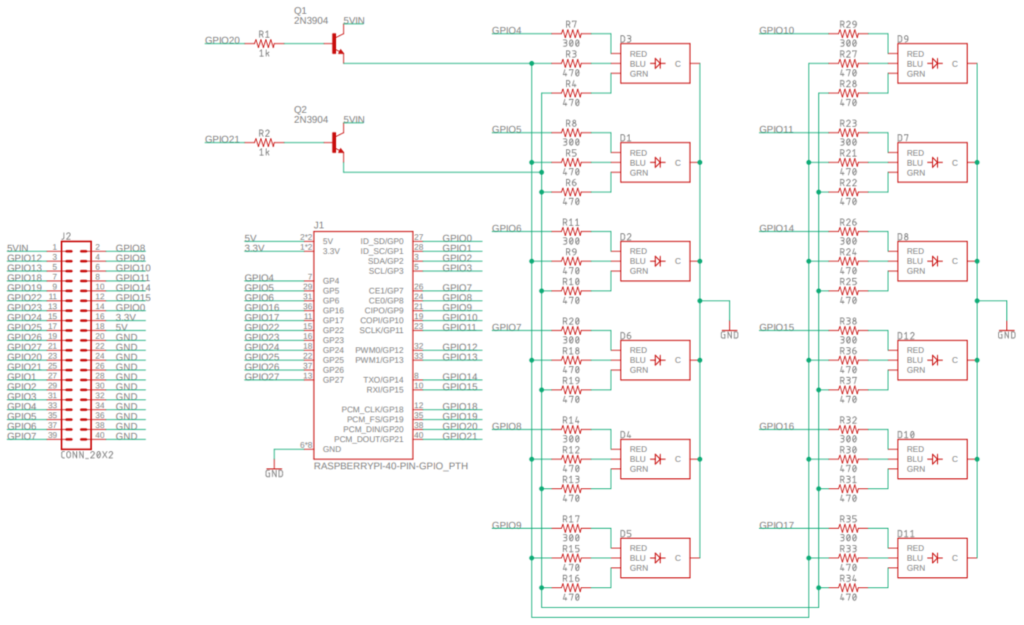


Figure 10: GPIO PCB Schematic, Array of LEDs

The GPIO board itself contains complex routing due to the number of connections needed. This can be seen in figure 11.

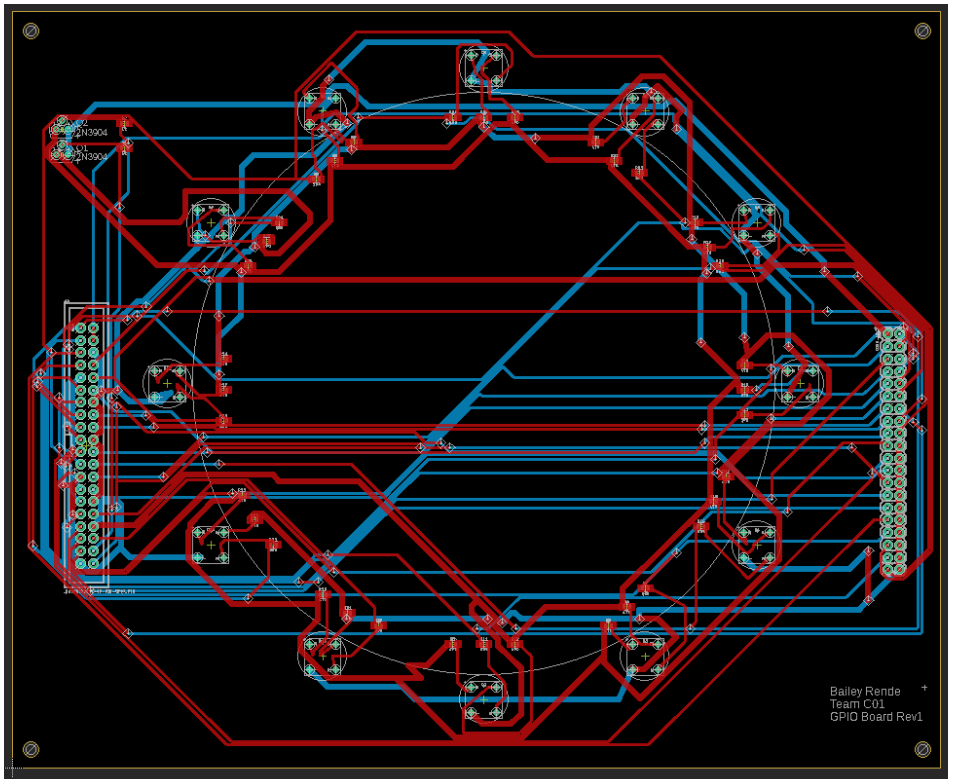


Figure 11: GPIO PCB Connections

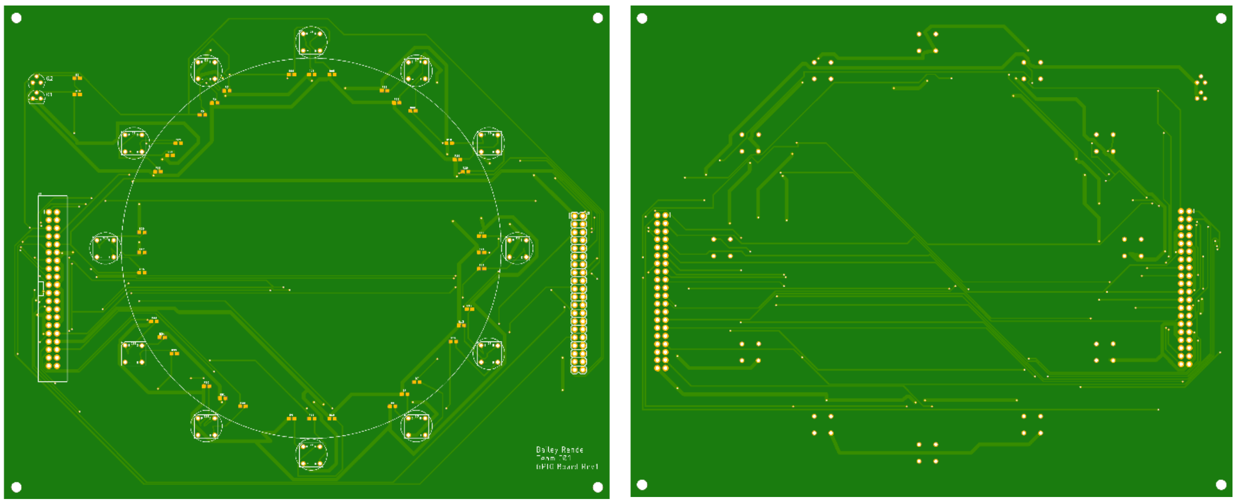


Figure 12: GPIO PCB top (left) and bottom (right) view

Thick traces are required to handle the possible current demand of the Pi 4 and the servos at maximum demand. Two bucks are required to eliminate the possibility of exceeding their ratings if the system is pulling maximum power. This is reflected in the power PCB schematic shown below.

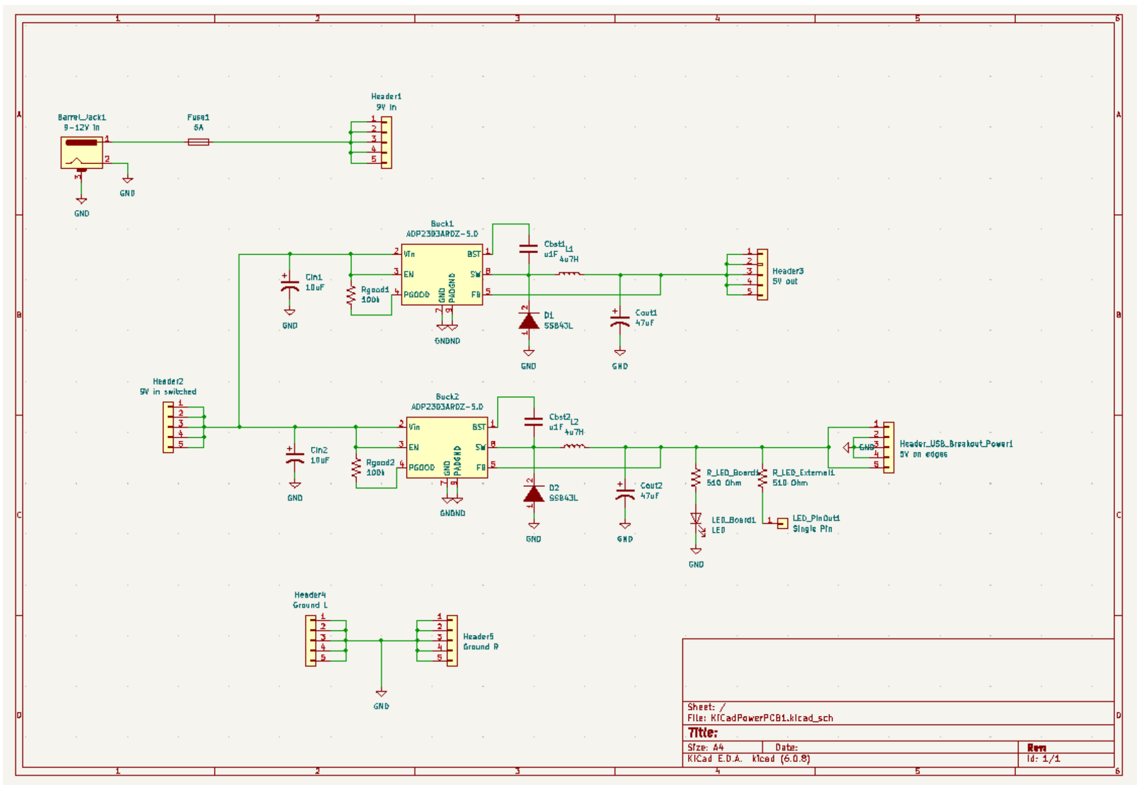


Figure 13: Power PCB Schematic

The power PCB itself is much easier to follow than the GPIO board due to it containing fewer components and not having a high density pin location such as the GPIO headers. It could have been manufactured as a single sided PCB, however it requires a ground plane to absorb and radiate the thermal generation of the buck converters. As such the reverse side is a single large ground plane.

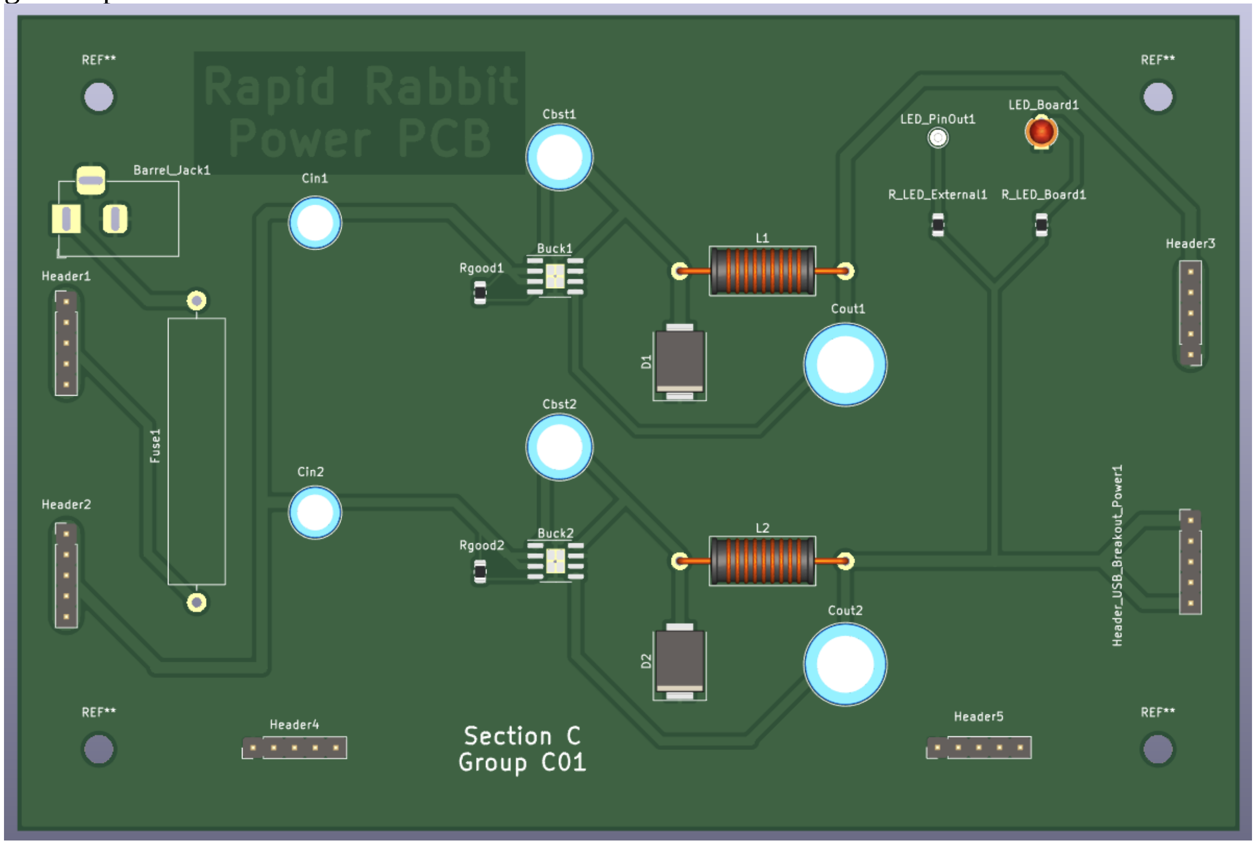


*Figure 14: Power PCB Model with Components*

## Electronic Simulation

Power demands of individual components were collected, budgeted, and the power PCB is designed with these demands in mind. The traces of the power PCB are designed to carry the possible 3A maximum that the Pi could draw and a second buck converter supplies power to the servos and other LEDs to keep the current lower and within trace ratings.
The maximum calculated power draw of the system is 22.3W. With the 45W supply being used, there is significant room for error and losses without exceeding the limits of the supply or the project requirement of <60W.

## Mechanical Design

The mechanical design features a box with the rabbit sitting on top. As seen in figure 15, the box is roughly be 12” in width, 5” in height, and 12” in length. The box is made of 1/8” wood that will be laser cut. The rabbit that is sitting on top will be a store-bought stuffed animal. The front of the box will include various switches, buttons, and LEDs as a user interface. It also includes a speaker grill and volume rocker.

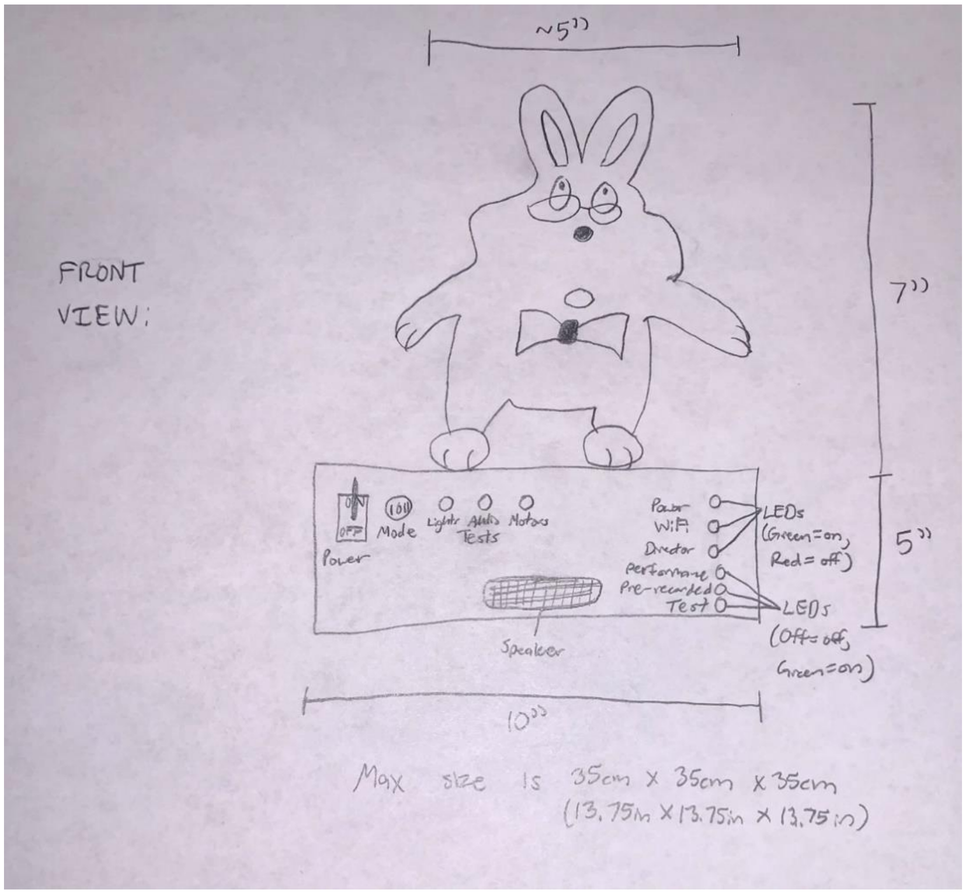


*Figure 15: Preliminary Front View Sketch of Robot Structure*

As seen in figure 16, the rear of the box has the power in barrel jack and an acrylic window. The window will allow for visibility inside and will have holes to allow for ventilation. The window will also be removable to allow for access inside.

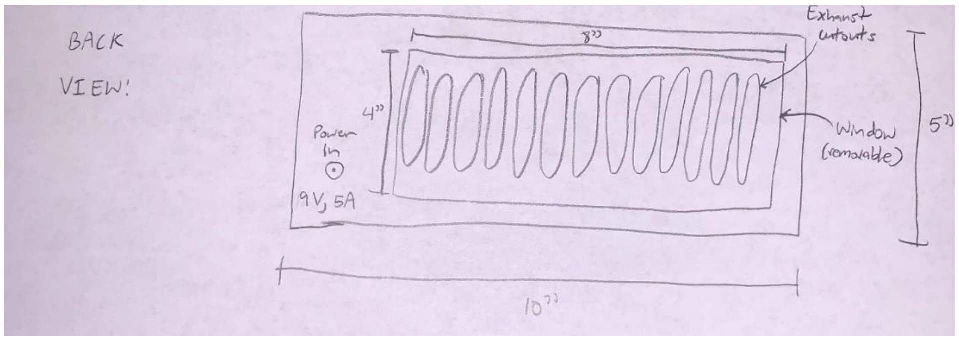


*Figure 16: Preliminary Back View Sketch of Box that Robot Stands On*

In addition to the rabbit being on the top of the box, a clock is also there. As seen in figure 17, the clock features 12 RGB LEDs, each one corresponding to an hour on the clock. The clock is roughly 5” in diameter and has an arbitrary time shown with an hour and minute hand. This sits roughly 1/2” in front of the rabbit.

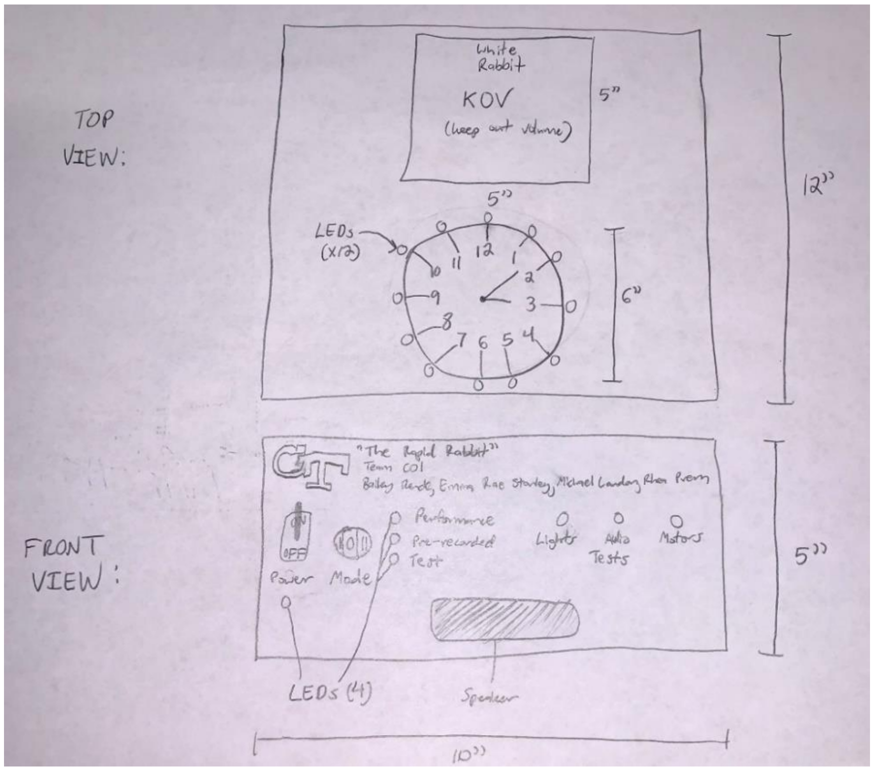


Figure 17: Top View Structural Design and Updated Front View

As seen in figure 18, the mechanical design also features a pair of servos that individually move the rabbit’s ear up and down. The rabbit has firm ears, so the motor simply needs to pull the ear down and release it for an up and down motion. To do this, the servo has an attachment with fishing line tied around it. The fishing line goes through the top of the box and attaches to the back of each ear.

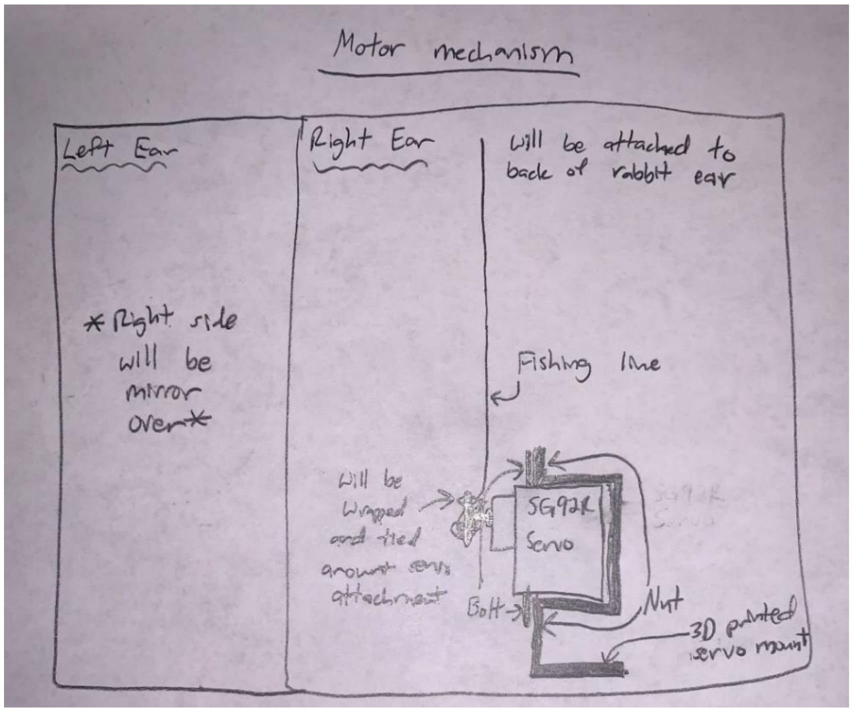

Motor mechanism
Left Ear
Right Ear
Will be attached to back of rabbit ear
*Right side will be mirror over*
Fishing line
Will be wrapped and tied around servo attachment
SG92R Servo
Bolt
Nut
3D printed servo mount

*Figure 18: Preliminary Sketch of Motor Mechanism Design*

To easily laser cut prototypes and the final design, the robot base was created in Inkscape. The files created were SVGs, with red indicating laser cuts black indicating laser engraving. This can be seen in figure 19. Even with the rabbit atop, the dimensions are well within the design constraints.

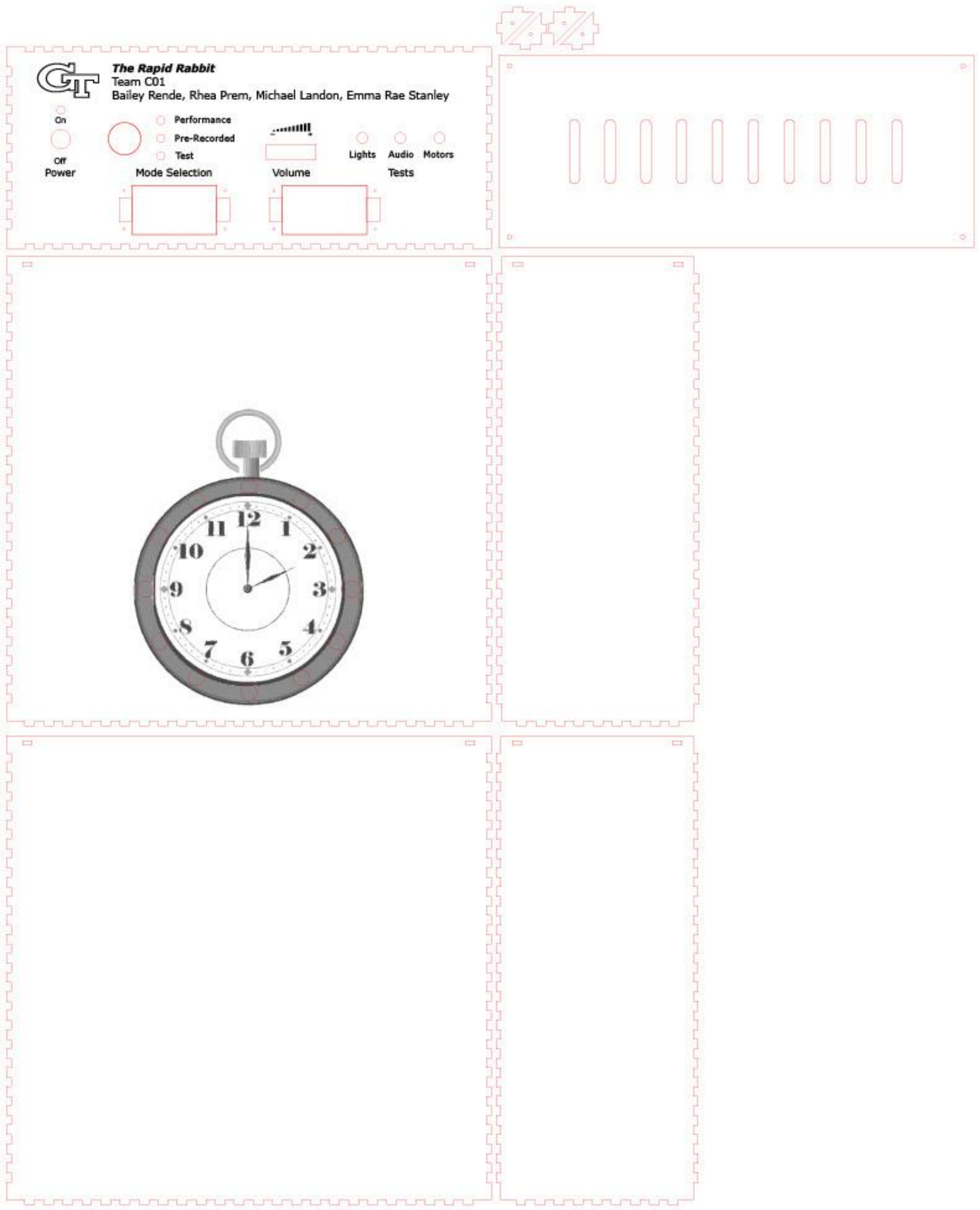


*Figure 19: SVG File of Robot Base Final Design*

To test the initial robot base design, cardboard was laser cut. After, it was assembled using hot glue. The only cutouts were on the back, which were for airflow and the DC power jack. Overall, the design came out nice and will be used as a base design.

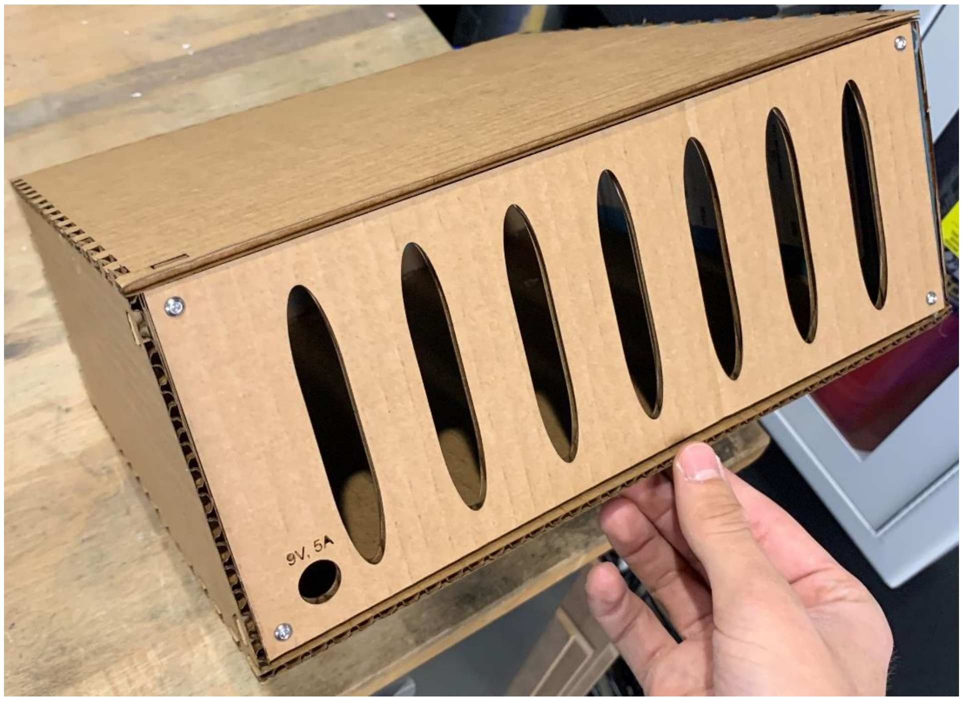


*Figure 20: Laser Cut Cardboard Prototype of Robot Base*

Next, cutouts, text, and images were added to the design. This includes a pocket watch, the GT logo, LED cutouts, speaker cutouts, cutouts for switches and buttons, and text of the project name, team, and team members. To test this design, wood was laser cut/engraved. Some of the text and cutouts got misaligned, and some cutouts were missing. Despite this, the overall design was great.

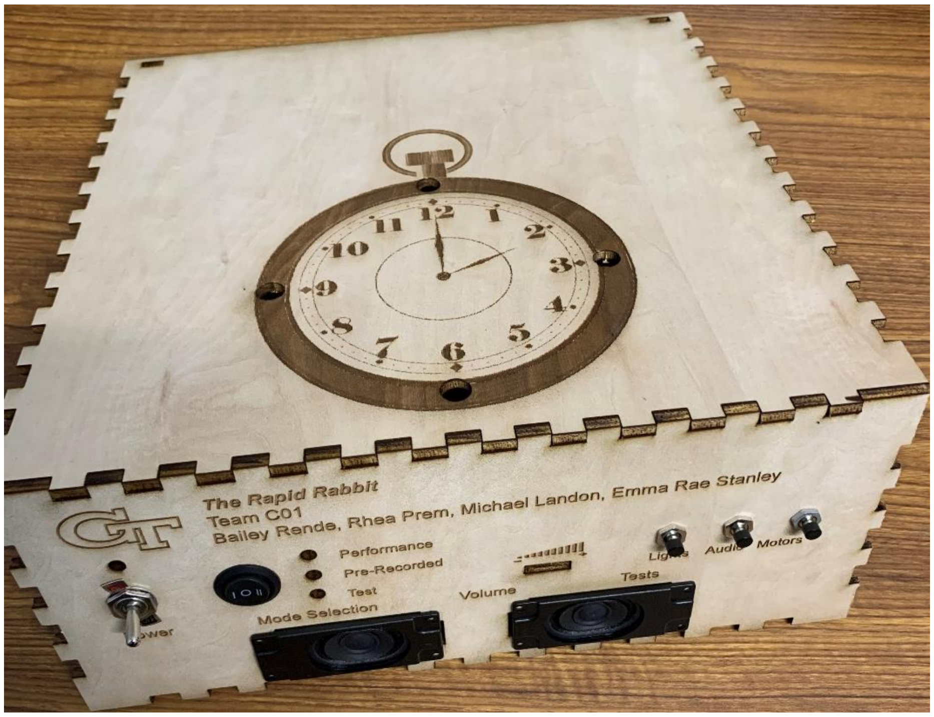


*Figure 21: Laser Cut Wood Prototype of Robot Base*

To address the final issues, a final design was created and laser cut. The results show great functionally and aesthetics. This can be seen below in figure 22.

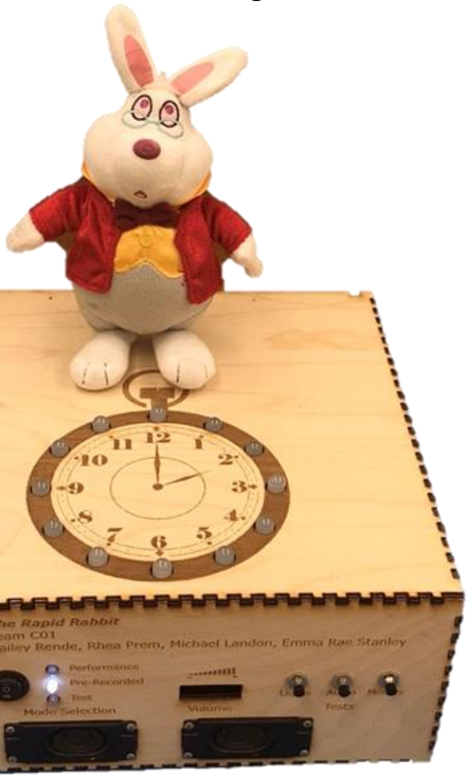


*Figure 22: Laser Cut Wood of Robot Base Final Design*

To securly mount the rabbit to the box, two holes were cut on the top piece of the box. Then, two wooden dowels were put through the hole and glued in place. Finally, the rabbit's feet were cut open to allow for the dowels to slide through. The design can be seen below in figure 23.

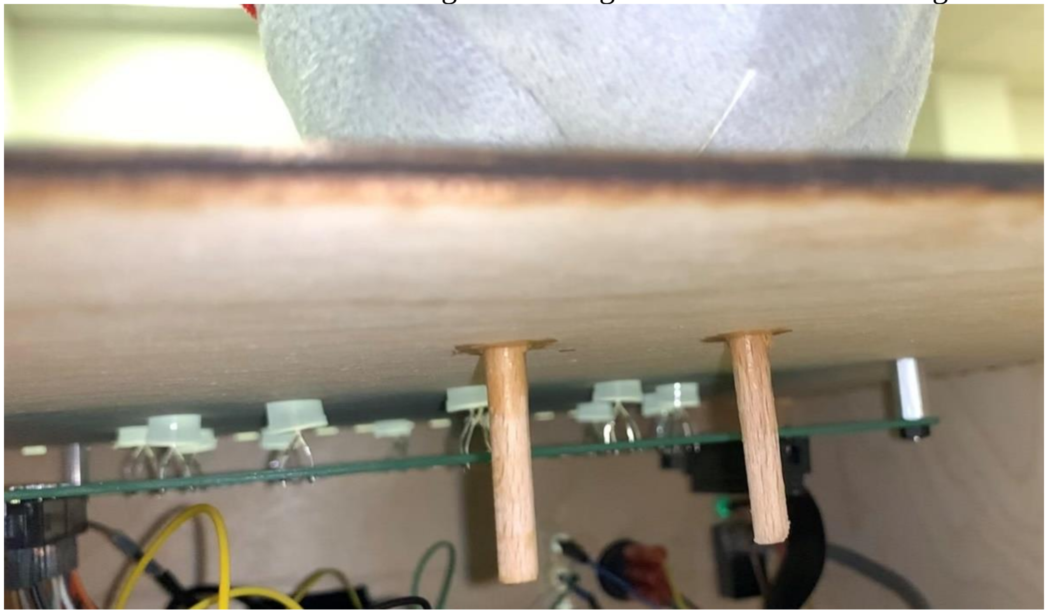


*Figure 23: Wooden Dowels to Secure Rabbit to Box*

To secure the servo motor inside of the box and to make sure that it does not come loose or get in the way of other components, a motor mount was designed. This motor mount can be seen below in figure 23.

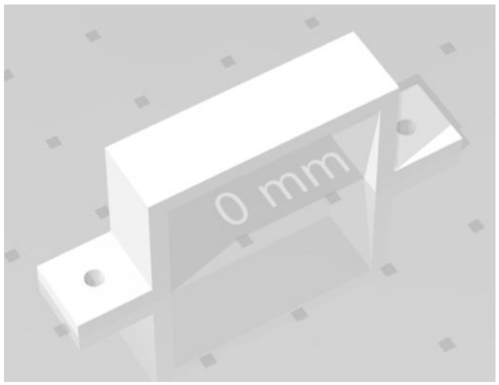


*Figure 24: Motor Mount Designed in SolidWorks*

The TowerPro SG90 comes with several arms, which we've discovered are not large enough. Custom ones were designed to overcome this.

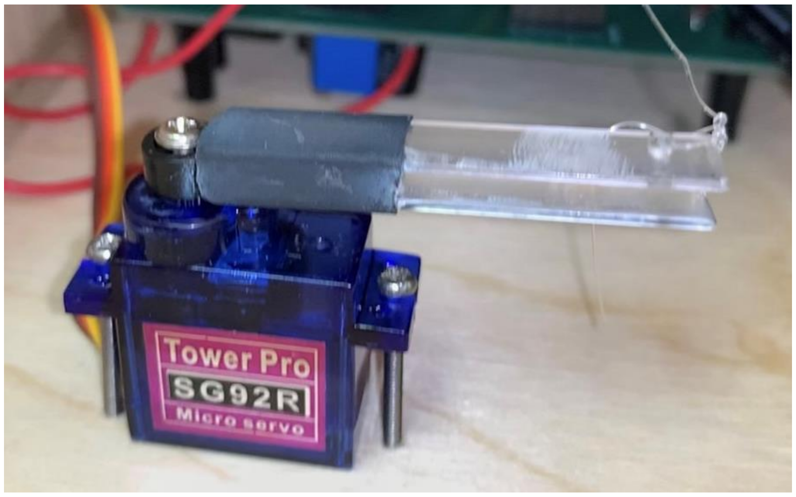


*Figure 25: TowerPro SG92R Servo with Custom Arm*

## Mechanical Simulation

One aspect of simulating the mechanical design was testing how the rabbit's ear, thread, and servo motor interact together to make the ear move. Initially, the ears were not as stiff as they were assumed to be, so reinforcements had to be made. This was accomplished by making a shape that roughly matched the ear out of wire and then on the rabbit, sewing that piece of wire onto the back of the rabbit ear. This gave the rabbit ear more structure so that it could bend down and rise back up properly. The wire sewn into the back of the rabbit ear can be seen in figure 25 below.

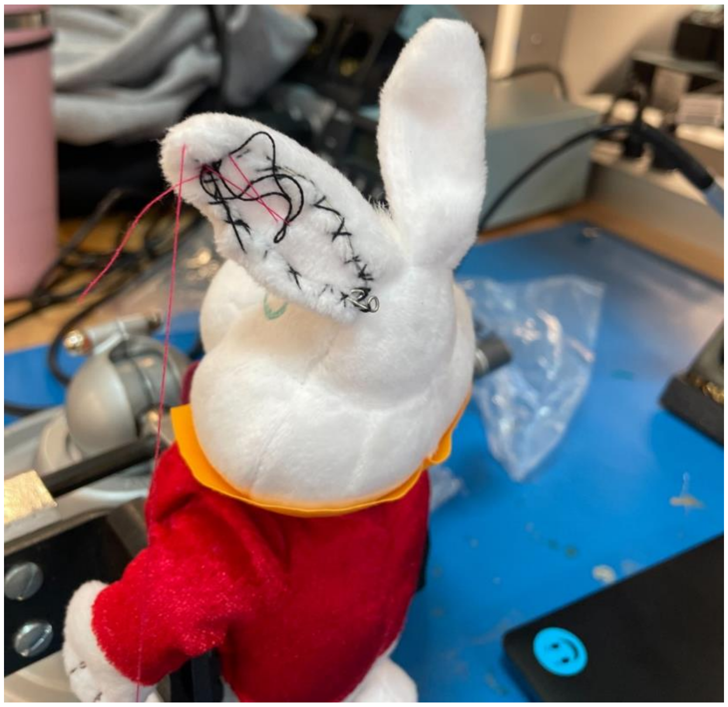


*Figure 26: Wire Sewn into Ear to Reinforce it*

The next step in simulating the motion was moving the servo arm back and forth 180 degrees through code. For this test, thread was attached at the top of the ear and the end of the servo arm. This setup can be seen in figure 26 below.

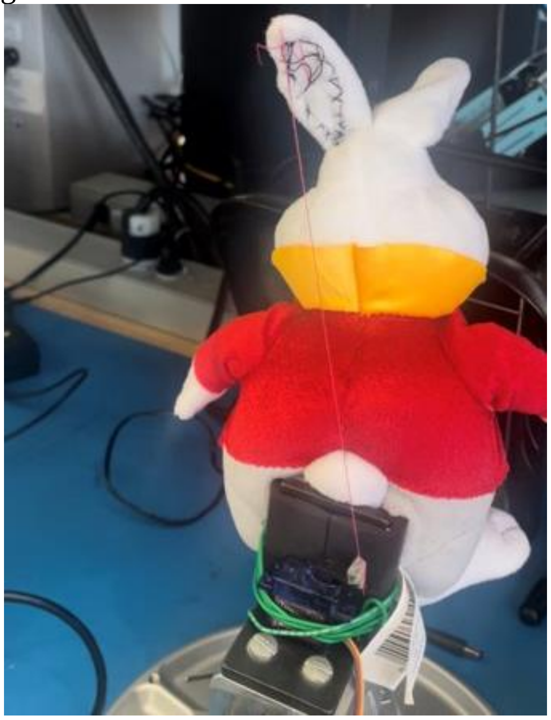


*Figure 27: Thread Attached to Ear to Move Up and Down via Servo*

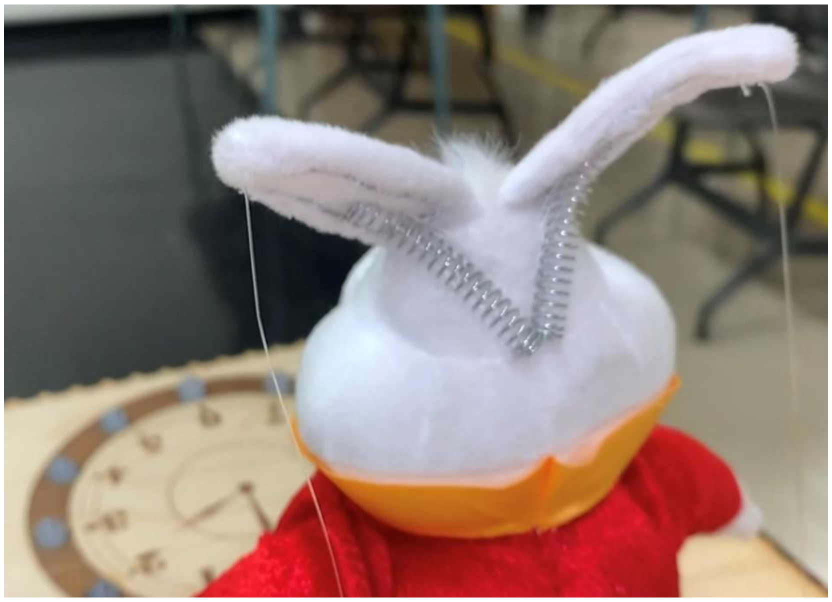


*Figure 28: Final Ear Design with a Spring and Fishing Line*

All of this results in the knowledge that the ear can be moved up and down with the servo in this configuration. However, a longer servo arm will be required to get a bigger range of motion. Finally, fishing line will be used from now on for aesthetic purposes.

## Software Design

The software design for our system includes 8 states: start, testing mode, pre-recorded mode, connect to director, line received, wait, perform, and end. From the initial start state, there is a three-mode switch on the front of our root that will give input on what the next state is. We are basing our software architecture off these states. In testing mode, which is the middle position of our three-mode switch, we will have individual switches to test our motor, audio, and lights. The top position of the three-mode switch is pre-recorded mode. This will have a predetermined/preprogrammed set of movements and features to perform in case the robot fails to connect to the director. Finally, the bottom position of the three-mode switch will attempt to connect to the director and upon successful connection the robot will perform.

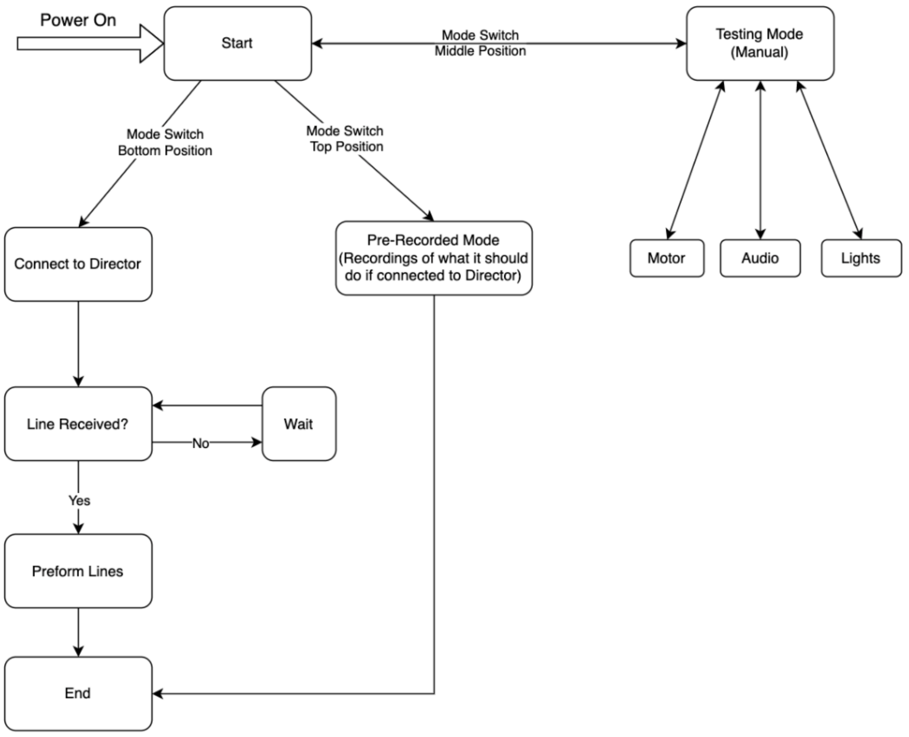


*Figure 29: Software State Machine with 11 States.*

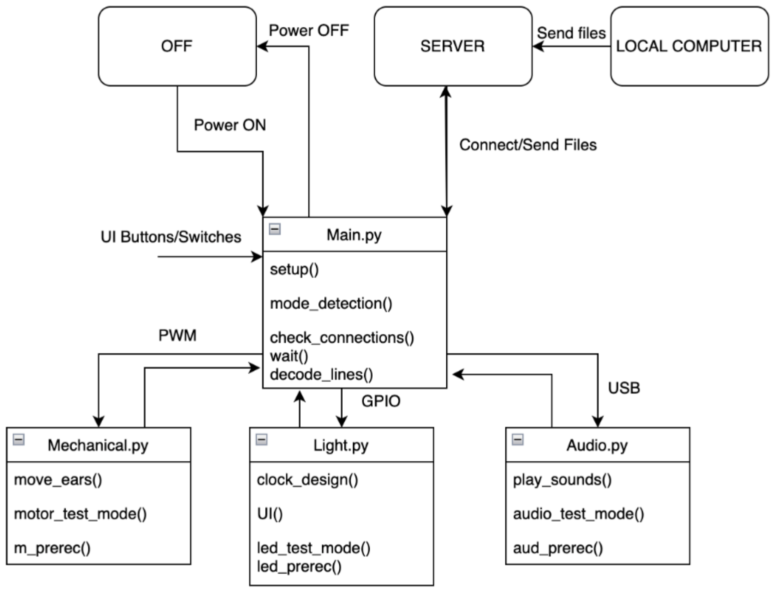


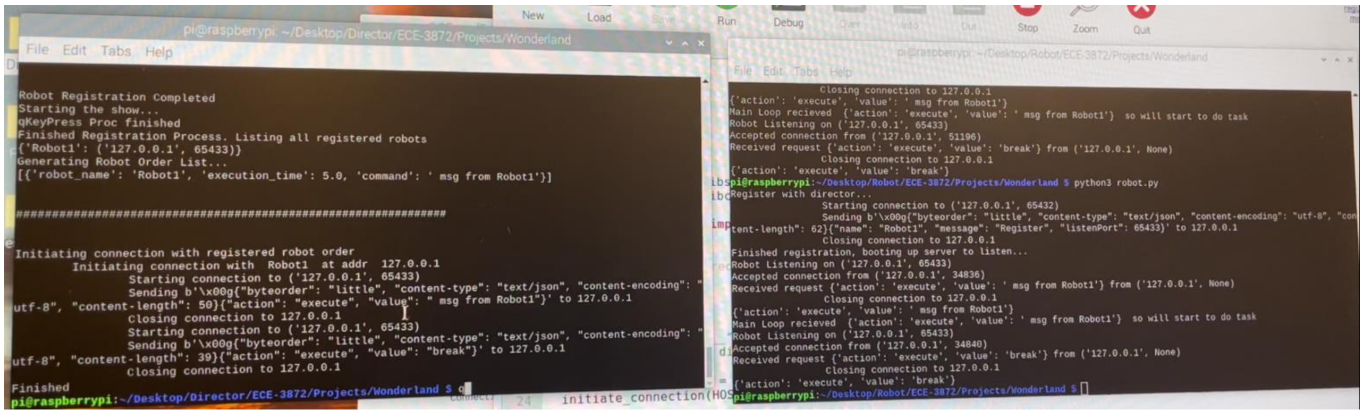
*Figure 30: Software Architecture*

Above is a diagram of our software architecture. Main.py will be the primary loop for our program, it consists of functions such as setup(), which will setup gpio pin assignments and other initializations, mode_detection(), which will take in input from our mode selection switch and determine which mode to move into, check_connections(), which will see if the robot is connected to the director yet, wait(), which is the idle state as we wait for connection to the director, and decode_lines(), which will decode the lines the director sends. The other sub-system files, Light.py, Audio.py, and Mechanical.py, have similar layouts to each other. There are test mode functions such as motor_test_mode(), prerecorded mode functions such as led_prerec(), and live mode functions such as play_sounds().

## Software Simulation

The first iteration of the software simulation we performed was connecting and receiving default messages from the director. Here is a link to a recorded version of the simulation.

In this simulation, we are running the director script, or director.py, with a specified IP address to mimic the real director we will be connecting to on performance day. Next, we are running our robot's script, or robot.py, and establishing connection to the director. If this is successfully done, we should see the robot registered in the director terminal and in our robot terminal we will receive the messages put in the director's CSV file.



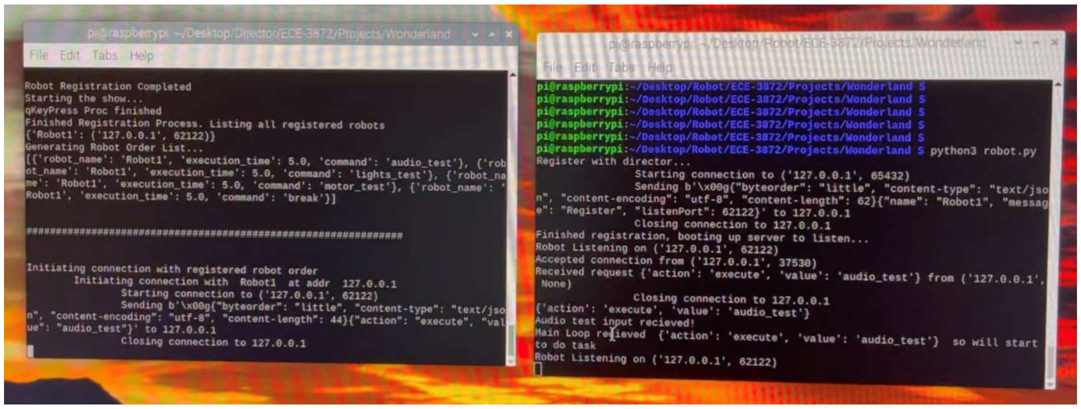
```
pi@raspberrypi: ~/Desktop/Director/ECE-3872/Projects/Wonderland

Robot Registration Completed
Starting the show...
qKeyPress Proc finished
Finished Registration Process. Listing all registered robots
{'Robot1': ('127.0.0.1', 65433)}
Generating Robot Order List...
[{'robot_name': 'Robot1', 'execution_time': 5.0, 'command': ' msg from Robot1'}]

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


Initiating connection with registered robot order
        Initiating connection with  Robot1  at addr  127.0.0.1
                Starting connection to ('127.0.0.1', 65433)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/json", "content-encoding": "
utf-8", "content-length": 50}{"action": "execute", "value": " msg from Robot1"}' 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", "content-encoding": "
utf-8", "content-length": 39}{"action": "execute", "value": "break"}' to 127.0.0.1
                Closing connection to 127.0.0.1

Finished
pi@raspberrypi:~/Desktop/Director/ECE-3872/Projects/Wonderland $ q
```

```
pi@raspberrypi: ~/Desktop/Robot/ECE-3872/Projects/Wonderland

                Closing connection to 127.0.0.1
{'action': 'execute', 'value': ' msg from Robot1'}
Main Loop recieved  {'action': 'execute', 'value': ' msg from Robot1'}  so will start to do task
Robot Listening on ('127.0.0.1', 65433)
Accepted connection from ('127.0.0.1', 51196)
Received request {'action': 'execute', 'value': 'break'} from ('127.0.0.1', None)
                Closing connection to 127.0.0.1
{'action': 'execute', 'value': 'break'}
pi@raspberrypi:~/Desktop/Robot/ECE-3872/Projects/Wonderland $ python3 robot.py
Register with director...
                Starting connection to ('127.0.0.1', 65432)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/json", "content-encoding": "utf-8", "con
tent-length": 62}{"name": "Robot1", "message": "Register", "listenPort": 65433}' to 127.0.0.1
                Closing connection to 127.0.0.1
Finished registration, booting up server to listen...
Robot Listening on ('127.0.0.1', 65433)
Accepted connection from ('127.0.0.1', 34836)
Received request {'action': 'execute', 'value': ' msg from Robot1'} from ('127.0.0.1', None)
                Closing connection to 127.0.0.1
{'action': 'execute', 'value': ' msg from Robot1'}
Main Loop recieved  {'action': 'execute', 'value': ' msg from Robot1'}  so will start to do task
Robot Listening on ('127.0.0.1', 65433)
Accepted connection from ('127.0.0.1', 34840)
Received request {'action': 'execute', 'value': 'break'} from ('127.0.0.1', None)
                Closing connection to 127.0.0.1
{'action': 'execute', 'value': 'break'}
pi@raspberrypi:~/Desktop/Robot/ECE-3872/Projects/Wonderland $
```

Figure 31: Image of Initial Successful Connection to the Director

Above are the results from our software simulation. The leftmost terminal window is where we are running the director script while the rightmost is running our robot script. On the left window, we can see clear indications that we have connected successfully to the director. For example, we see "Robot Registration Completed", "Initiating connection with registered robot order", and "Sending b'\x00g...". These are all print statements in director.py that signify successful connection to the director. If further proof is needed, in the right window, we can see more print statements that show successful connection and communication. For example, "Finished registration, booting up server to listen...", "Accepted connection from {'127.0.0.1',34836}", and "Main loop received {'action': 'execute', 'value': 'msg from Robot1'} so will start to do task". These are print statements in robot.py that signify successful connection to the director. In particular, the most important print statement we see is "Main loop received {'action': 'execute', 'value': 'msg from Robot1'} so will start to do task". This shows that the director read the CSV file, found the value we put in, and sent the value 'msg from Robot1' to our robot!

As a follow-up to this original simulation, we also performed a second simulation as an extension of our first. In this simulation, we are not only ensuring that the connection between our robot and the director is sound but also receiving multiple values from the director.

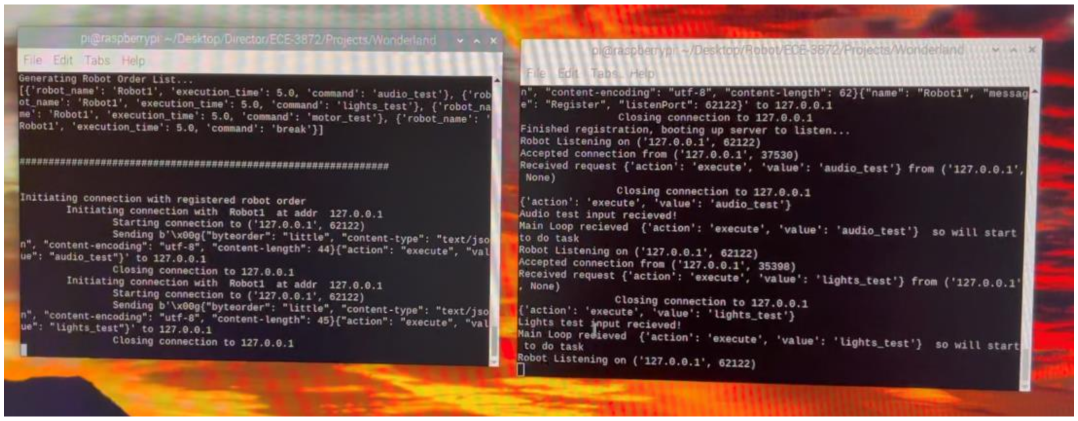


```
pi@raspberrypi: ~/Desktop/Director/ECE-3872/Projects/Wonderland

Robot Registration Completed
Starting the show...
qKeyPress Proc finished
Finished Registration Process. Listing all registered robots
{'Robot1': ('127.0.0.1', 62122)}
Generating Robot Order List...
[{'robot_name': 'Robot1', 'execution_time': 5.0, 'command': 'audio_test'}, {'rob
ot_name': 'Robot1', 'execution_time': 5.0, 'command': 'lights_test'}, {'robot_na
me': 'Robot1', 'execution_time': 5.0, 'command': 'motor_test'}, {'robot_name': '
Robot1', 'execution_time': 5.0, 'command': 'break'}]

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


Initiating connection with registered robot order
        Initiating connection with  Robot1  at addr  127.0.0.1
                Starting connection to ('127.0.0.1', 62122)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/jso
n", "content-encoding": "utf-8", "content-length": 44}{"action": "execute", "val
ue": "audio_test"}' to 127.0.0.1
                Closing connection to 127.0.0.1
```

```
pi@raspberrypi: ~/Desktop/Robot/ECE-3872/Projects/Wonderland

pi@raspberrypi:~/Desktop/Robot/ECE-3872/Projects/Wonderland $
pi@raspberrypi:~/Desktop/Robot/ECE-3872/Projects/Wonderland $
pi@raspberrypi:~/Desktop/Robot/ECE-3872/Projects/Wonderland $
pi@raspberrypi:~/Desktop/Robot/ECE-3872/Projects/Wonderland $
pi@raspberrypi:~/Desktop/Robot/ECE-3872/Projects/Wonderland $
pi@raspberrypi:~/Desktop/Robot/ECE-3872/Projects/Wonderland $ python3 robot.py
Register with director...
                Starting connection to ('127.0.0.1', 65432)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/jso
n", "content-encoding": "utf-8", "content-length": 62}{"name": "Robot1", "messag
e": "Register", "listenPort": 62122}' to 127.0.0.1
                Closing connection to 127.0.0.1
Finished registration, booting up server to listen...
Robot Listening on ('127.0.0.1', 62122)
Accepted connection from ('127.0.0.1', 37530)
Received request {'action': 'execute', 'value': 'audio_test'} from ('127.0.0.1',
 None)

                Closing connection to 127.0.0.1
{'action': 'execute', 'value': 'audio_test'}
Audio test input recieved!
Main Loop recieved  {'action': 'execute', 'value': 'audio_test'}  so will start
to do task
Robot Listening on ('127.0.0.1', 62122)
```

Figure 32: Second Iteration of Software Simulation with Successful Initiation of Audio Test

Here the Pi is receiving a value which tells it to start the audio sequence.



pi@raspberrypi: ~/Desktop/Director/ECE-3872/Projects/Wonderland
File Edit Tabs Help

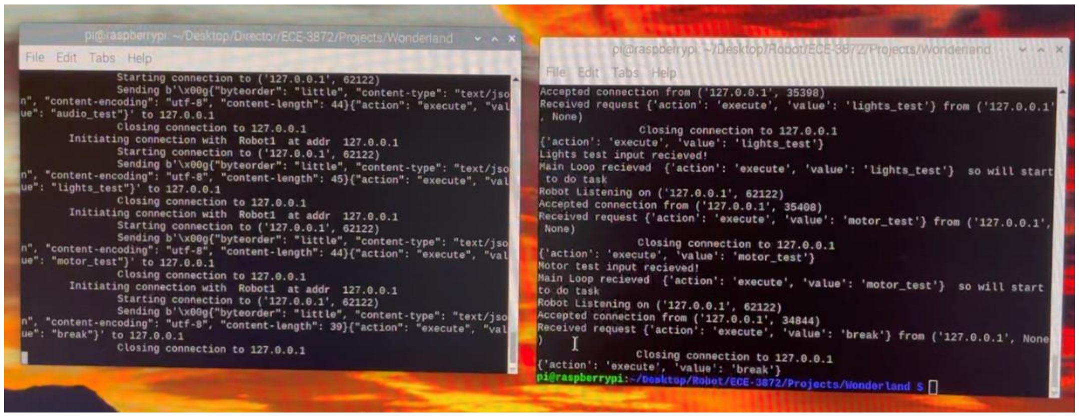
```
Generating Robot Order List...
[{'robot_name': 'Robot1', 'execution_time': 5.0, 'command': 'audio_test'}, {'rob
ot_name': 'Robot1', 'execution_time': 5.0, 'command': 'lights_test'}, {'robot_na
me': 'Robot1', 'execution_time': 5.0, 'command': 'motor_test'}, {'robot_name': '
Robot1', 'execution_time': 5.0, 'command': 'break'}]


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


Initiating connection with registered robot order
        Initiating connection with  Robot1  at addr  127.0.0.1
                Starting connection to ('127.0.0.1', 62122)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/jso
n", "content-encoding": "utf-8", "content-length": 44}{"action": "execute", "val
ue": "audio_test"}' to 127.0.0.1
                Closing connection to 127.0.0.1
        Initiating connection with  Robot1  at addr  127.0.0.1
                Starting connection to ('127.0.0.1', 62122)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/jso
n", "content-encoding": "utf-8", "content-length": 45}{"action": "execute", "val
ue": "lights_test"}' to 127.0.0.1
                Closing connection to 127.0.0.1
```

pi@raspberrypi: ~/Desktop/Robot/ECE-3872/Projects/Wonderland
File Edit Tabs Help

```
n", "content-encoding": "utf-8", "content-length": 62}{"name": "Robot1", "messag
e": "Register", "listenPort": 62122}' to 127.0.0.1
                Closing connection to 127.0.0.1
Finished registration, booting up server to listen...
Robot Listening on ('127.0.0.1', 62122)
Accepted connection from ('127.0.0.1', 37530)
Received request {'action': 'execute', 'value': 'audio_test'} from ('127.0.0.1',
 None)
                Closing connection to 127.0.0.1
{'action': 'execute', 'value': 'audio_test'}
Audio test input recieved!
Main Loop recieved  {'action': 'execute', 'value': 'audio_test'}  so will start
to do task
Robot Listening on ('127.0.0.1', 62122)
Accepted connection from ('127.0.0.1', 35398)
Received request {'action': 'execute', 'value': 'lights_test'} from ('127.0.0.1'
, None)
                Closing connection to 127.0.0.1
{'action': 'execute', 'value': 'lights_test'}
Lights test input recieved!
Main Loop recieved  {'action': 'execute', 'value': 'lights_test'}  so will start
 to do task
Robot Listening on ('127.0.0.1', 62122)
```

Figure 33: Successful Receival of Lights Test Command from Director

Next, the Pi receives a value to start the lights sequence.



pi@raspberrypi: ~/Desktop/Director/ECE-3872/Projects/Wonderland
File Edit Tabs Help

```
                Starting connection to ('127.0.0.1', 62122)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/jso
n", "content-encoding": "utf-8", "content-length": 44}{"action": "execute", "val
ue": "audio_test"}' to 127.0.0.1
                Closing connection to 127.0.0.1
        Initiating connection with  Robot1  at addr  127.0.0.1
                Starting connection to ('127.0.0.1', 62122)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/jso
n", "content-encoding": "utf-8", "content-length": 45}{"action": "execute", "val
ue": "lights_test"}' to 127.0.0.1
                Closing connection to 127.0.0.1
        Initiating connection with  Robot1  at addr  127.0.0.1
                Starting connection to ('127.0.0.1', 62122)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/jso
n", "content-encoding": "utf-8", "content-length": 44}{"action": "execute", "val
ue": "motor_test"}' to 127.0.0.1
                Closing connection to 127.0.0.1
        Initiating connection with  Robot1  at addr  127.0.0.1
                Starting connection to ('127.0.0.1', 62122)
                Sending b'\x00g{"byteorder": "little", "content-type": "text/jso
n", "content-encoding": "utf-8", "content-length": 39}{"action": "execute", "val
ue": "break"}' to 127.0.0.1
                Closing connection to 127.0.0.1
```

pi@raspberrypi: ~/Desktop/Robot/ECE-3872/Projects/Wonderland
File Edit Tabs Help

```
Accepted connection from ('127.0.0.1', 35398)
Received request {'action': 'execute', 'value': 'lights_test'} from ('127.0.0.1'
, None)
                Closing connection to 127.0.0.1
{'action': 'execute', 'value': 'lights_test'}
Lights test input recieved!
Main Loop recieved  {'action': 'execute', 'value': 'lights_test'}  so will start
 to do task
Robot Listening on ('127.0.0.1', 62122)
Accepted connection from ('127.0.0.1', 35408)
Received request {'action': 'execute', 'value': 'motor_test'} from ('127.0.0.1',
 None)
                Closing connection to 127.0.0.1
{'action': 'execute', 'value': 'motor_test'}
Motor test input recieved!
Main Loop recieved  {'action': 'execute', 'value': 'motor_test'}  so will start
to do task
Robot Listening on ('127.0.0.1', 62122)
Accepted connection from ('127.0.0.1', 34844)
Received request {'action': 'execute', 'value': 'break'} from ('127.0.0.1', None
)
                Closing connection to 127.0.0.1
{'action': 'execute', 'value': 'break'}
pi@raspberrypi:~/Desktop/Robot/ECE-3872/Projects/Wonderland $
```

Figure 34: Successful Receival of Motor Test Command from Director

And finally, the Pi receives a value to start the motor sequence and then break.

## Schedule

The schedule for completion of the project is shown in Table 3. Below in Table 3 is a list of task leads in charge of each sub-system.

*Table 3: Schedule to Complete Rapid Rabbit*

Name: Team C01
Project: *The Rapid Rabbit*

| | Week Number | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Task** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** |
| Brainstorm | Emma | | | | | | | | | | | | |
| Design Top Level Block Diagram | Michael | | | | | | | | | | | | |
| Design Software Block Diagram | Rhea | | | | | | | | | | | | |
| **Proposal** | | ■ | | | | | | | | | | | |
| **Audio Design** | | | | | | | | | | | | | |
| Find micro USB speaker, layout framework | | | Bailey | Bailey | | | | | | | | | |
| Determine volume knob integration | | | | Bailey | Bailey | | | | | | | | |
| **Lighting Design** | | | | | | | | | | | | | |
| Rough draft/ideas | | Michael | Michael | | | | | | | | | | |
| Test different RGB LEDs | | | Michael | Michael | | | | | | | | | |
| Design circuit | | | | Michael | | | | | | | | | |
| **Mechanical Design** | | | | | | | | | | | | | |
| Motor capability research | | | Emma | Emma | | | | | | | | | |
| Sketch rough draft of component interactions | | | | Emma | Emma | | | | | | | | |
| Preliminary design to hold motors | | | | | Emma | Emma | | | | | | | |
| Design electronics box/base to laser cut | | | Bailey | Bailey | Bailey | | | | | | | | |
| Find rabbit stuffed animal | | | | Bailey | | | | | | | | | |
| **Software Controller Design** | | | | | | | | | | | | | |
| Function Hierarchy Design | | | | Rhea | | | | | | | | | |
| Initial CSV File | | | | Rhea | | | | | | | | | |
| **Power Design** | | | | | | | | | | | | | |
| Power circuit design | | | | Michael | Michael | Michael | | | | | | | |
| **PDR** | | | | | ■ | | | | | | | | |
| **Audio Design Revision** | | | | | | | | | | | | | |
| Simulate audio by file playback on USB speaker | | | | | | Bailey | | | | | | | |
| **Lighting Design Revision** | | | | | | | | | | | | | |
| Lighting simulation software for clock display | | | | | | Michael | | | | | | | |
| Lights PCB model and order | | | | | | | Bailey | Bailey | | | | | |
| **Mechanical Design Revision** | | | | | | | | | | | | | |
| Battery mount design and printing | | | | | | | Emma | | | | | | |
| Component interaction simulation with cardboard and servos | | | | | | | | Emma | | | | | |
| Reinforcing rabbit ears | | | | | | | | Emma | | | | | |
| 3D Printing servo arm extenders | | | | | | | | | Emma | | | | |
| Laser cut cardboard and assemble prototype design | | | | | | Bailey | | | | | | | |
| Revise electronics box/base design | | | | | | Bailey | Bailey | | | | | | |
| **Software Controller Design Revision** | | | | | | | | | | | | | |
| Simulate inputs and outputs to other subsystems | | | | | | Rhea | | | | | | | |
| Connect to director | | | | | | | Rhea | | | | | | |
| **Power Design Revision** | | | | | | | | | | | | | |
| Determine power budget | | | | | Michael | Michael | | | | | | | |
| Make revisions to specs if needed | | | | | | | Michael | | | | | | |
| Power PCB modeling and order | | | | | | | Michael | Michael | | | | | |
| **CDR** | | | | | | | | | ■ | | | | |
| **Audio Build** | | | | | | | | | | | | | |
| Install speakers and volume knob in housing | | | | | | | | | | Bailey | | | |
| Connect speaker to Pi and save final sound files | | | | | | | | | | | Bailey | | |
| **Lighting Build** | | | | | | | | | | | | | |
| Lights PCB soldering | | | | | | | | | | Bailey | | | |
| Perform preliminary lights PCB tests and install | | | | | | | | | | Bailey | Bailey | | |
| **Mechanical Build** | | | | | | | | | | | | | |
| Laser cut wood and assemble design | | | | | | | | | Bailey | Bailey | | | |
| Attach UI buttons/switches to housing | | | | | | | | | Bailey | | | | |
| Motor mount build | | | | | | | | | Emma | Emma | | | |
| Attached servo arm to rabbit ear with fishing line | | | | | | | | | | Emma | | | |
| Attach motor/motor mount to box | | | | | | | | | | Emma | Emma | | |
| **Software Controller Build** | | | | | | | | | | | | | |
| Create lights.py, motor.py, and audio.py | | | | | | | Rhea | Rhea | | | | | |
| Preliminary test code for each subsystem | | | | | | | | Rhea | Rhea | | | | |
| Integration of more subsystems with code | | | | | | | | | | Rhea | | | |
| **Power Build** | | | | | | | | | | | | | |
| Power PCB soldering | | | | | | | | | Michael | Michael | | | |
| **Audio Test** | | | | | | | | | | | | | |
| Play sounds and adjust volume levels | | | | | | | | | | | Bailey | | |
| **Lighting Test** | | | | | | | | | | | | | |
| Confirm lighting effects run via lights test button | | | | | | | | | | Michael | Michael | | |
| **Mechanical Test** | | | | | | | | | | | | | |
| Confirm housing meets design requirements | | | | | | | | | | Bailey | Bailey | | |
| Check that all UI operates as expected | | | | | | | | | | | Bailey | | |
| Confirm there is a secure connection between motor and ear | | | | | | | | | | Emma | | | |
| Confirm that rabbit ear moves up and down with motor | | | | | | | | | | | Emma | | |
| Confirm motor mounting does not move during operation | | | | | | | | | | | Emma | | |
| **Software Controller Test** | | | | | | | | | | | | | |
| Run and test individual motor, lights, audio.py | | | | | | | | Rhea | | | | | |
| Connection to on campus director, editing CSV file | | | | | | | | | Rhea | Rhea | | | |
| Integrate .py files with hardware components (buttons, PCB, etc.) | | | | | | | | | | Rhea | Rhea | | |
| **Power Test** | | | | | | | | | | | | | |
| Power system with all lights on and motors running | | | | | | | | | | | Michael | | |
| **System Integration** | | | | | | | | | | | | | |
| | | | | | | | | | | Emma | | | |
| **System Test** | | | | | | | | | | | | | |
| | | | | | | | | | | | | Bailey | |
| **Final Inspection and Demonstration** | | | | | | | | | | | | ■ | |

| | | | |
|---|---|---|---|
| **Milestones** | Emma | Mechanical | |
| Tasks & Task Leads | Rhea | Software Controller | |
| | Bailey | Audio | Mechanical |
| | Michael | Lighting | Power |

## Integration

Our system is heavily dependent on each sub-system communicating with each other properly to ensure the success of the entire robot. Thus, we are compiling a testing and simulation schedule that covers all sub-systems so we can mitigate any mistakes. We are each completing sub-systems and doing intermediate testing to make sure they are working properly so when they are integrated into the overall system it will be as smooth as possible. Breaking down the system in this manner allows us to make sure each sub-system design and simulation is robust.

Software High-Risk Parts:

- The Raspberry Pi holds a lot of weight in our overall system because it interacts with every other sub-system. If the Pi fails to communicate with our other sub-systems, our robot will be practically useless.
    - To make sure the Pi is as reliable as possible, we are going to heavily test the code and interactions between the Pi and other sub-systems. This involves two parts: individually testing the code to remove any bugs and then integrating with each sub-system one by one.
    - So far, we have done preliminary tests with the Pi interacting with different LEDs and push buttons to make sure initial code works properly.

Hardware High-Risk Parts:

- The buck converters remain untested as they are surface mount components with particularly small pads.
    - While we are within the rated specifications are using it in accordance with configurations in its data sheet, we have not been able to verify how much heat they generate under load.
- Build time crunch
    - Waiting for PCBs to arrive: If problems occur during build, there is little time to correct.
        - The GPIO PCB has nearly 40 surface mount resistors alone, which will be time consuming to solder

Mechanical High-Risk Parts:

- The servo jerks in the beginning of its movement, which causes the thread to be pulled very harshly. If this continuously happens, it could result in the thread snapping.
- The servo arm length is another area of concern because we want to ensure it has enough range of motion to fully pull the ear down. The arms included with the servos are rather small, so larger ones are being 3D printed.
- The robot base design is made of very thin wood, which can easily break if not handled with care.

Repository Management:

- We have a GitHub that includes systems files such as software code.
    - https://github.gatech.edu/rprem3/Project_Wonderland_C01
- Other files are shared in our central Microsoft Teams SharePoint folder

## Conclusion

In the span of 12 weeks, our team was able to design and build The Rapid Rabbit. We broke down our overall design into 5 different sub-systems each led by one of our team members. By doing so, we were able to split the work evenly and test individual aspects of the design before integrating them all together. We were faced with many bumps along the road including getting our Pi connected to Wi-Fi and the Director, manipulating the stuffed animal so that movement was more noticeable, and PCB design. If given this project again, there are some changes our team would implement. We would have tried to have our schedule more thorough and follow it more strictly. This would have helped alleviate some portions of crunch time and have more concrete goals to implement. We also would have tried to meet in the lab more often to collaborate while working on subsystems. This would have helped us get input from each other and catch everyone up on progress throughout the week, rather than solely during our meetings. Overall, we were able to overcome these struggles and produce a clean and functioning robot that we are all proud of.