Design Document

Project Wonderland

Team #06: Joonseo Kim, Erin Tan, Marc Kouyoumji, Woods Burton

Date: November 15, 2022

Table of Contents

Revision Record

Date	**Author**	**Comments**
Sept 16, 2022	Team	Document Created
Sept 25, 2022	Erin Tan	Software Documentation, system design, hierarchical design, and state machine diagram
Oct 15, 2022	Woods Burton	Added electrical section
Oct 16, 2022	Erin Tan	Added Appendix A – robot.py and software simulation video. Added pictures of the simulations in software
Nov 15, 2022	Erin Tan	[TODO] Change Appendix A to reflect final robot code (gpio2.py, robot.py) Finalized Software Decomposition description to reflect final project

Project Description

This document details the verification tests that will be used for Project Wonderland. Project Wonderland is a mechanical character that is capable of receiving a text input and outputting speech audio. The character is a duck whose bill extends and retracts. Additionally, the duck's eyes light up when it's talking. Additionally, the character will be contained in a box of size

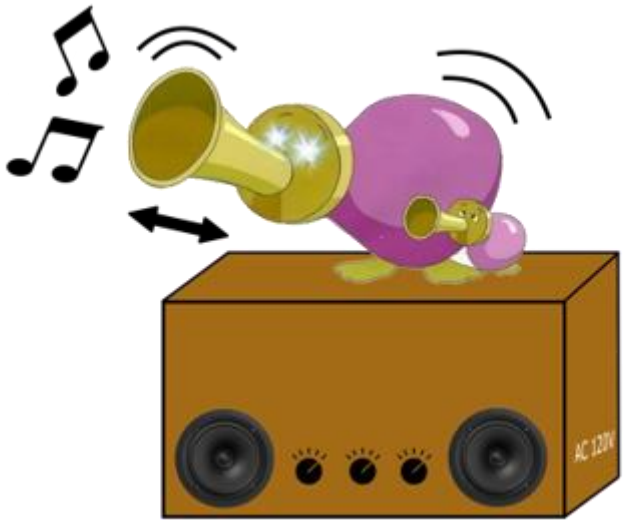


Figure 1: Sketch of Project Wonderland design.

System Design

Interface	Source	Destination	Description
Analog tones	User interface	Processor	Analog signal between 0V and 5V DC Analog signal from microphone
Digital Controls	User interface	Processor	5V DC signal for the 2 push buttons
Motor Control Signals	Processor	Motion	PWM signal to control to motors
PMW Audio	Processor	Audio	PMW signal to output the music
120V	System input	Power	120V AC input power
5V	Power	Processor Audio Motion	5V DC power 25W 5V DC power 10W 5V DC power 15W
3.3V	Processor	Motion	3.3V DC

Table 1: Sketch of Project Wonderland design.

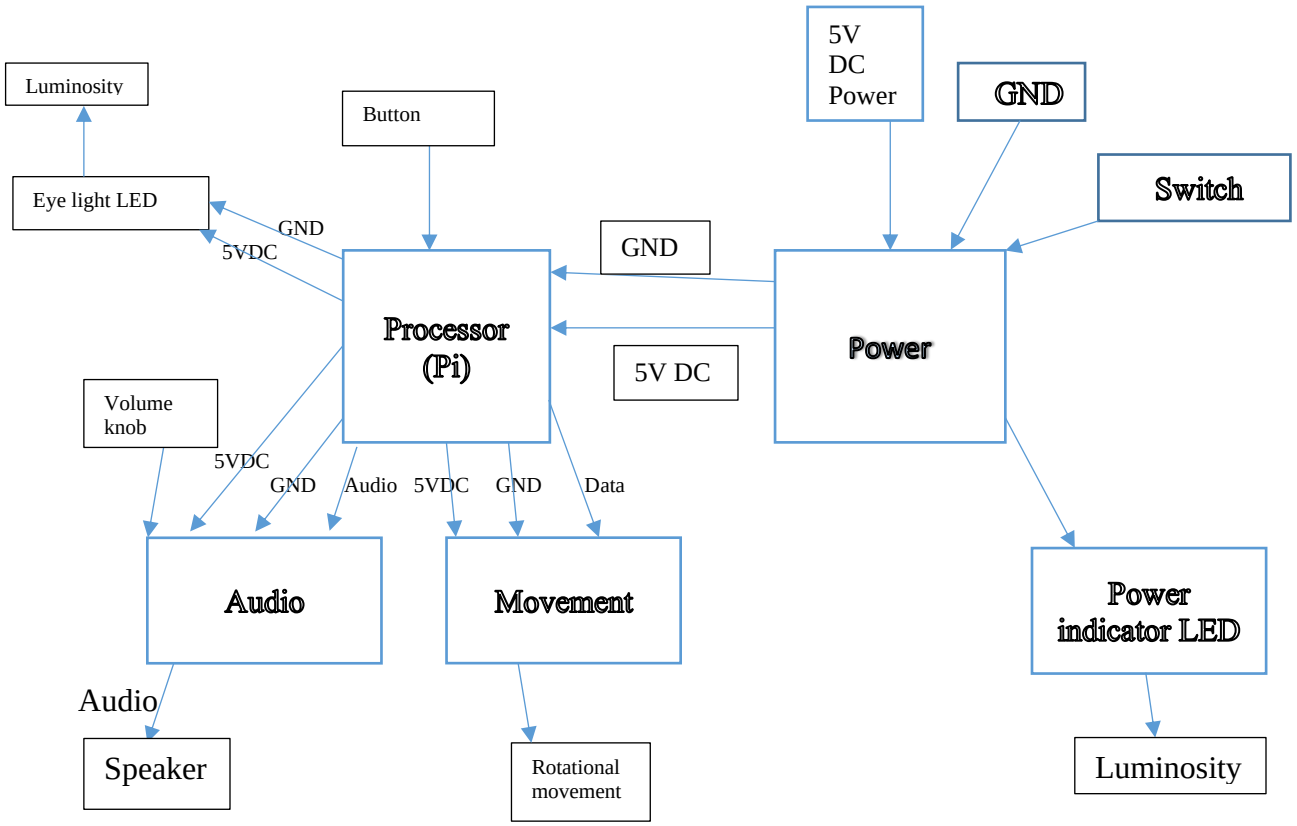


Figure 1: Sub-systems interfaces diagram

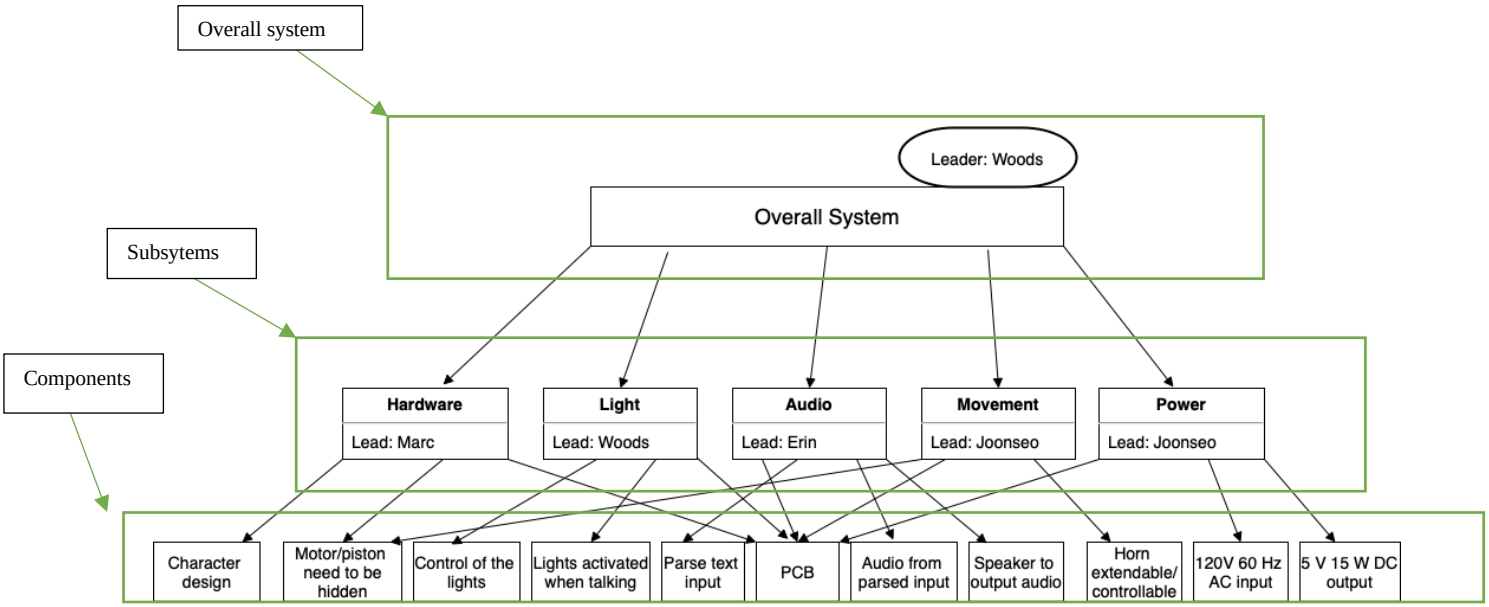


Figure 2: System hierarchy diagram demonstrating our subsystems of hardware, light, audio, movement, and power as well as their associated components.

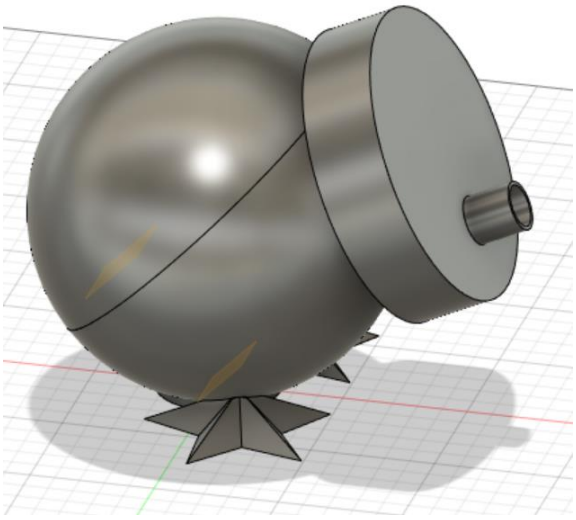
Sub-System Designs

User interface sub-system

The user interface has the following inputs: test button, volume knob and power switch.
Test button will run through each part of the system such as audio, light, and movement. Volume knob will use for changing the volume of sound output. Power switch will manually turn on and off the power input.

Hardware sub-system

For the hardware, a processor (PI) will take 5 Volts DC and Ground from the power supply. It will also be directed by a button. The processor will output 5V DC and GND to each of the eye light LED, audio sub-system, and movement sub-system. Additionally, the PI will provide the data for the movement sub-system as well as the necessary audio signal for the Audio sub-system. The character which will be 16cm x 20 cm x 17 cm will be sitting on top of a box of size 35 cm x 35 cm x 18 cm. The box model is generated with the ….. software



Audio sub-system

For the audio, the sub-system will take 5 Volts DC and Ground from the processor to the amplifier. Audio signal from processor will be connected to the volume knob, and after adjusting the strength of the signal by the user, it will connect to the amplifier. After that, amplifier will output an audio signal to the speaker.

Light sub-system

For the light, the sub-system will have two LEDs: one for the power indicator LED and one for the Eye Light LED. The power indicator LED will have power going in and coming from the power sub-system, and it will have luminosity that is outputted from the LED. The Eye Light LED will have the power going in and coming from the PI, and it will output luminosity.

Power sub-system

The objective of power sub-system is protecting system and circuit. Power source is 5V DC from the power adapter. It will connect to the system by 2.1mm DC barrel connector. After it connects to the system. it connected to the 10A fuse for circuit protection and power switch for user can on and off the system manually in series. When power switch is on, the power indicator light will light up while the power is being supplied. After that, the power sub-system will provide 5V DC to each sub-system.

Electronic Design

Our electrical design is centered around a Raspberry Pi 3 processor and a custom circuit board. The Raspberry Pi manages the entire system, and the circuit board enables the user to control and interact with the Pi to operate the device. An overview of the electrical connections within the system are shown below. The power supply, lights, speaker, and motor are standalone; all other components are contained in the circuit board. All outputs are direct from the Pi except for the audio signal which needs to be amplified prior to being sent to the speaker.

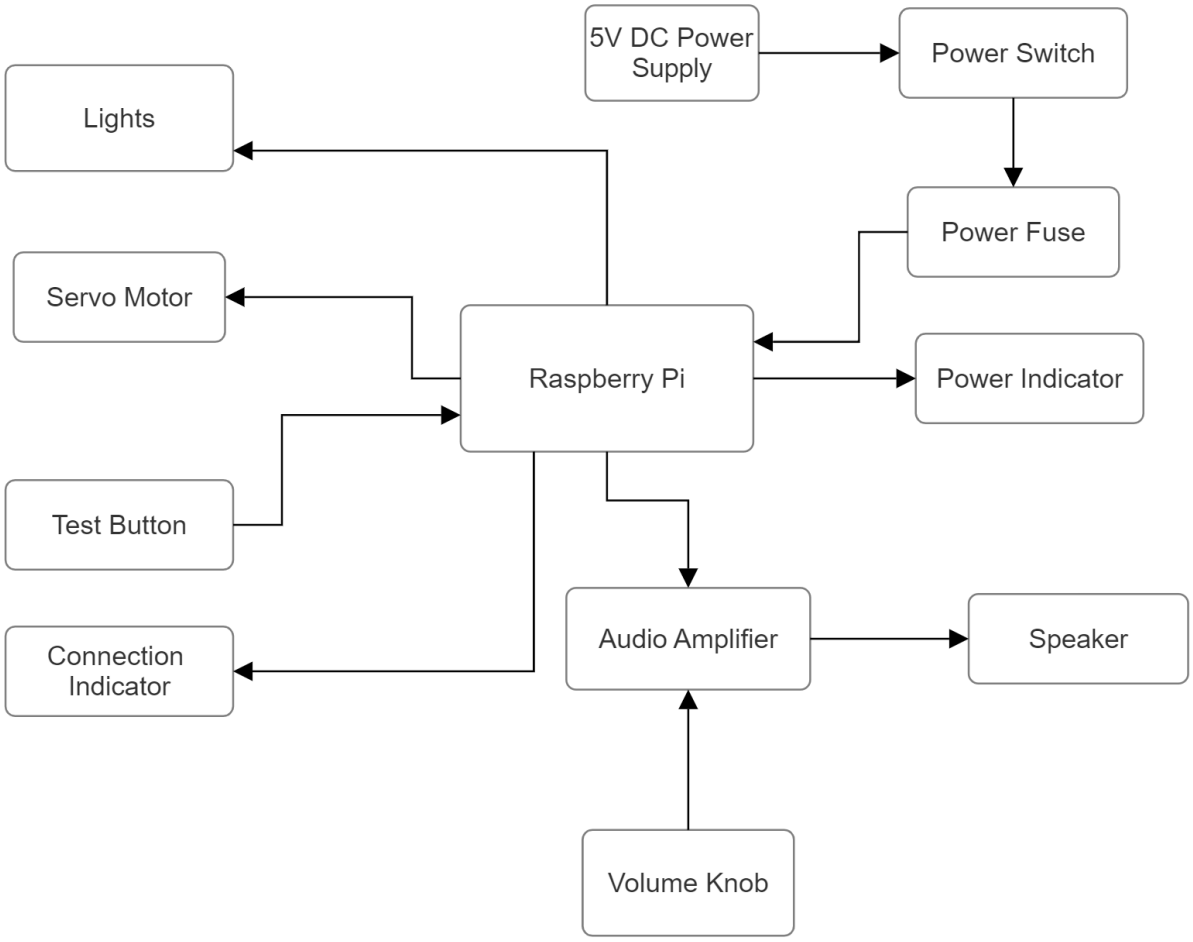


Figure 3: Electronic component block diagram

Created in Eagle, the schematic below shows the components and connections within the circuit board for the devices above. The circuit board contains four separate circuits: a power circuit, a connection indicator light, a test button, and an audio amplifier. There are three pin headers for connections to the Raspberry Pi, power supply, and speaker.

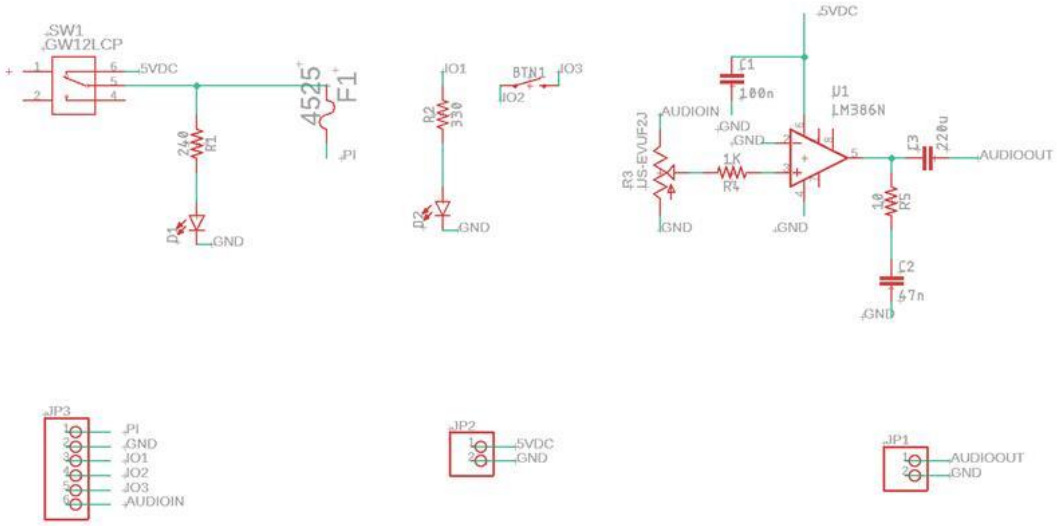


Figure 4: Schematic created in Eagle to connect and control the power supply, Raspberry Pi and speakers

The PCB was also created in Eagle using the schematic above. It is a two-layer PCB that contains our course section and group as well as labels for the components.

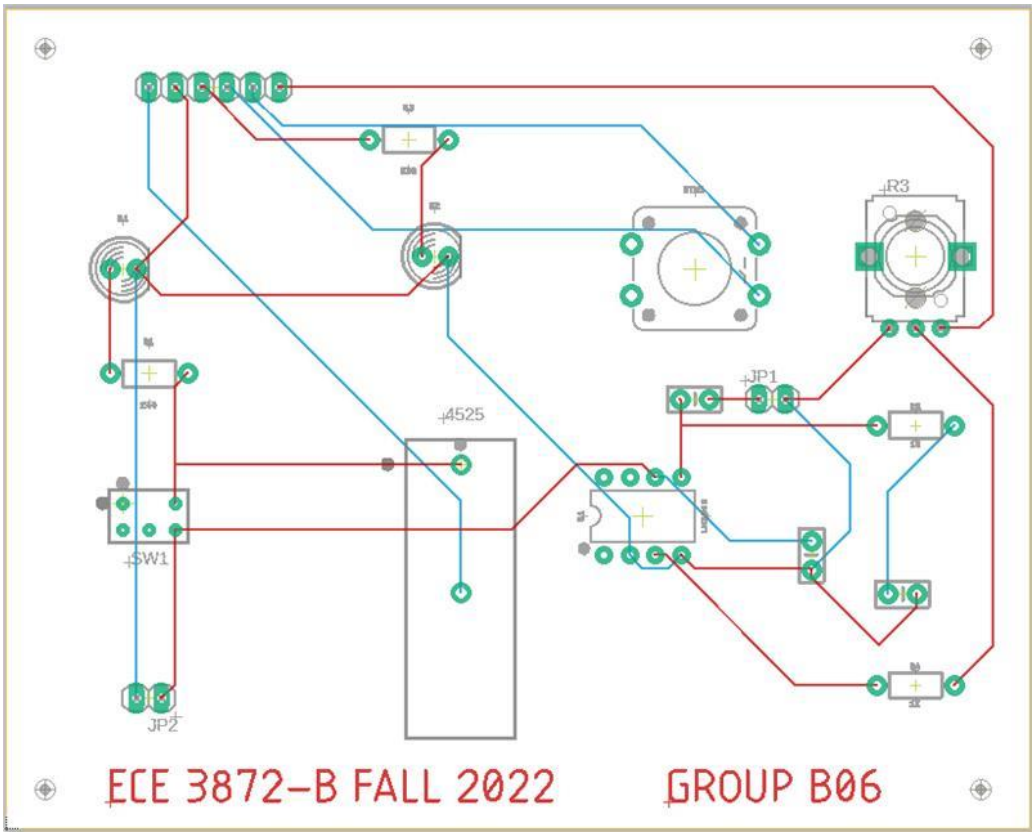


Figure 5: PCB created in Eagle based on the schematic in fig. 4

Movement Design

The movement system consists of a servo mounted inside the body of the duck. The servo is attached to a gear controlling a rack and pinion connected to the horn. This allows for the horn to move forward and backward in controlled manner.

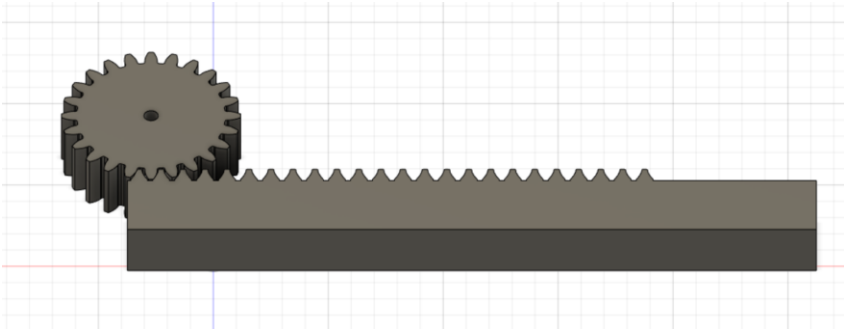


Figure 6: Rack and Pinion gear used for movement system in 3D-Design.

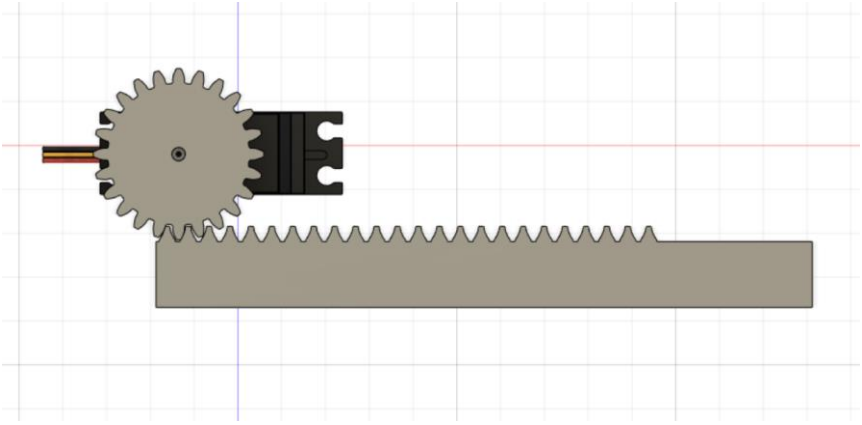


Figure 7 : Rack and Pinion gear on top-view with servo motor

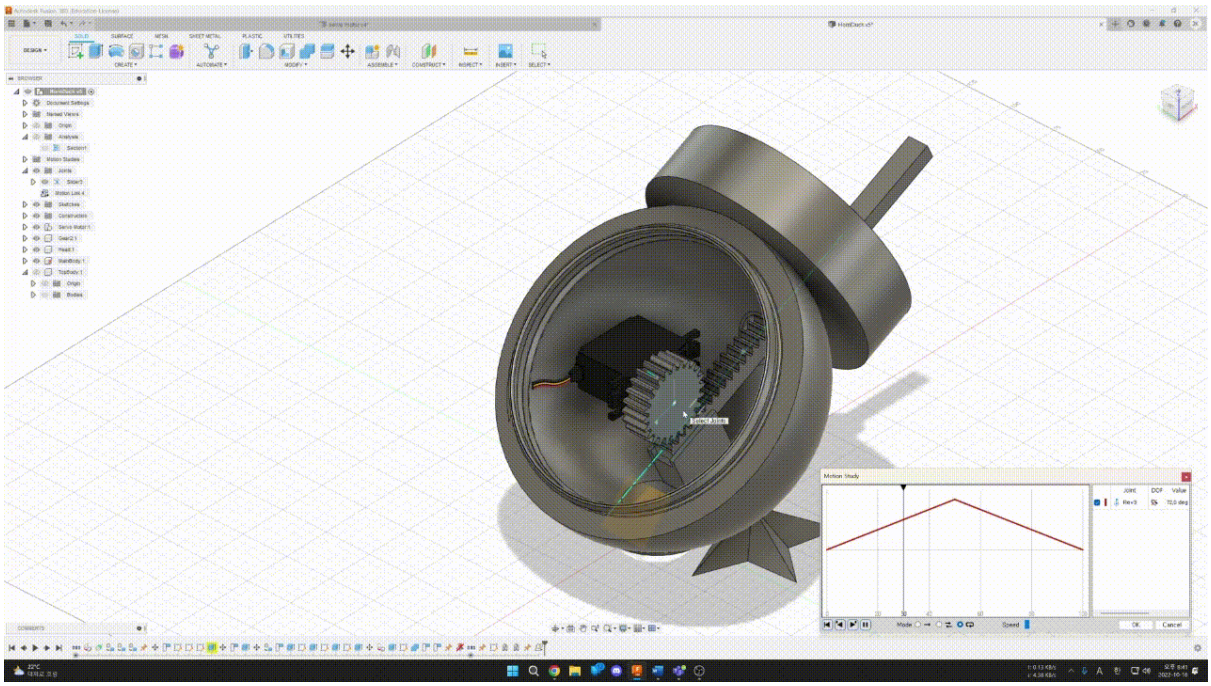


Figure 8 : Servo motor and Rack and Pinion gear inside of Character with simulating

Software Decomposition

Our software design includes 5 different states—off, idle, text-to-speech, output, and clear input as shown in Figure 9. Defining these states helped us lay out our software architecture as shown in the UML diagram in Figure 10. Using these two diagrams, we wrote code to emulate our design along with the motors, speaker, switches/test button, and Raspberry Pi defined in our schematic. The source code for the GPIO (General Purpose Input/Output) encoding and robot code to connect to the director and execute the functions can be found in Appendix A and B.

After writing the software code in the given robot.py file, several simulations were done to ensure the operation between the director and the robot. One of the simulations is demonstrated in the Software_Simulation_Video_B06.mp4 file in this folder. A test.csv file was given outlining the format of the final script, and the director/robot relationship was shown to be similar to a server/client relationship. As a result, the first simulation done was the successful registering of the robot with the director. Another simulation done was the robot being able to parse the test file and send it to an audio output. The last simulation done was having the Raspberry Pi be able to output the sound from this mp3 file.

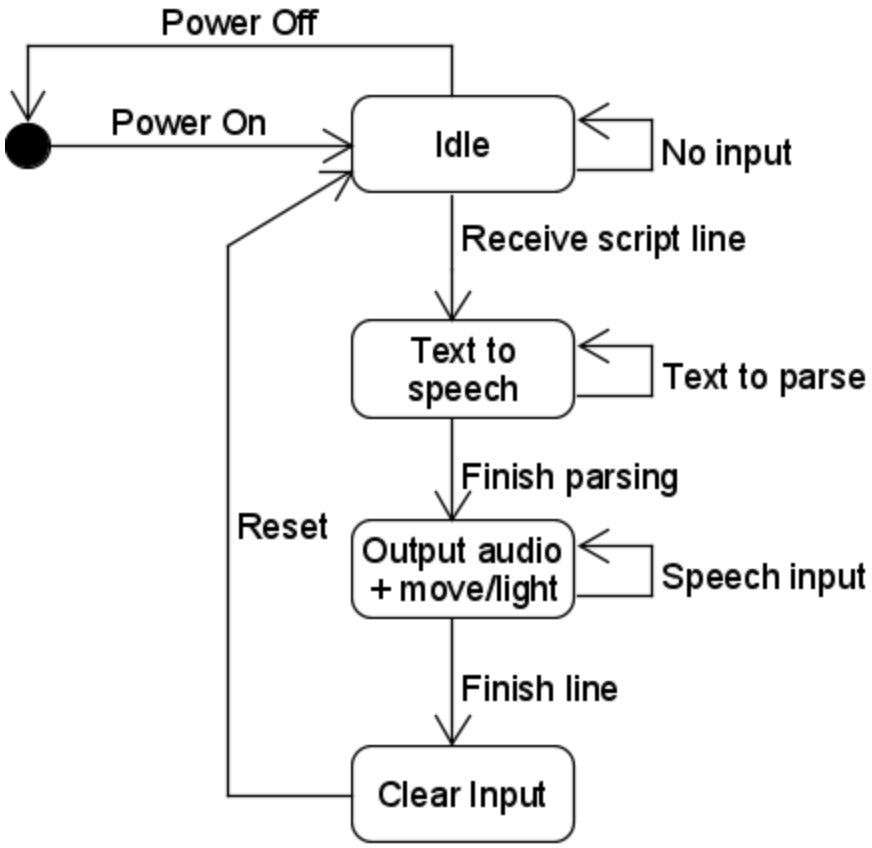


Figure 9: Software state machine with states off, idle, text-to-speech, output, and clear input.

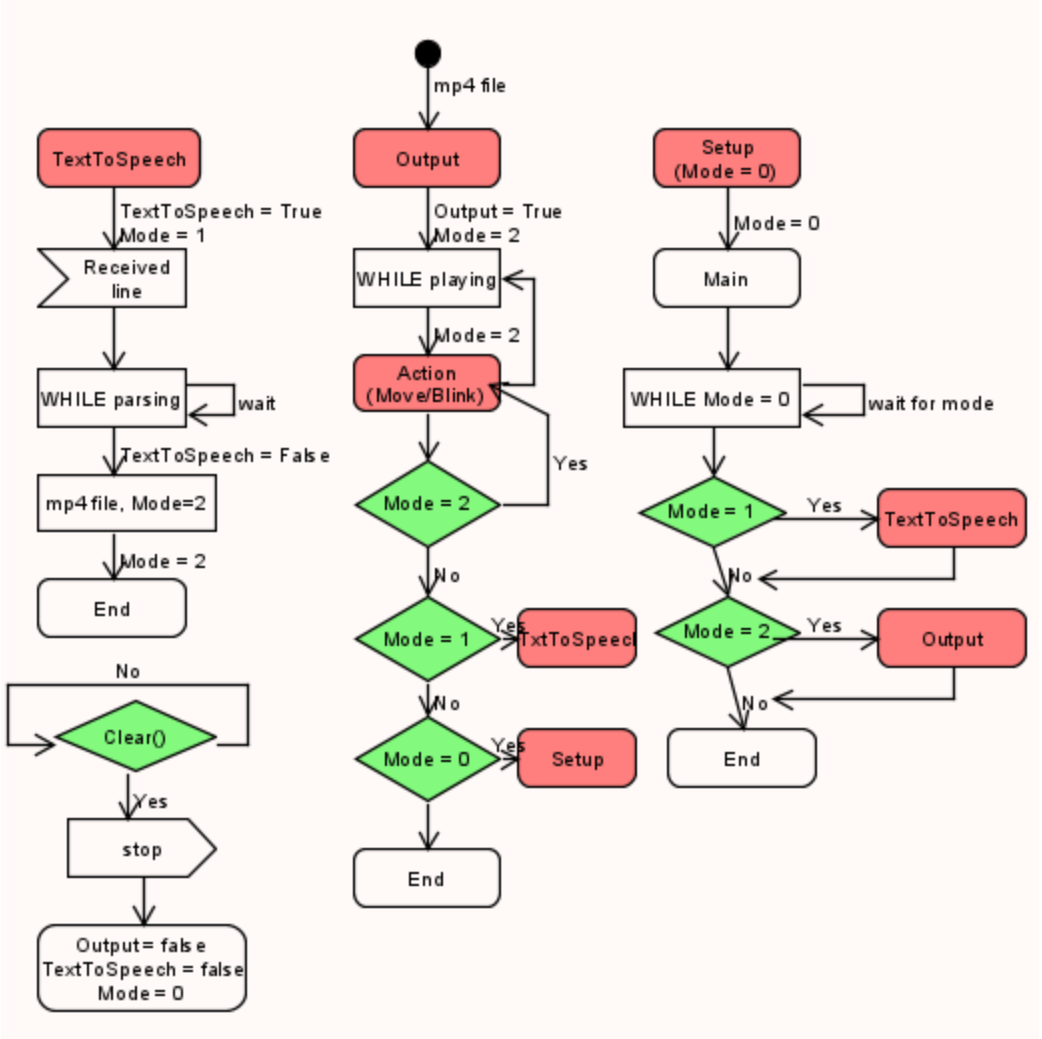


Figure 10: Flowchart detailing the software architecture that includes text-to-speech, action/output, and setup functions.

```
pi@raspberrypi:~/ECE-3872/Projects/Wonderland $ sudo python3 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', 59580)
Received request {'name': 'Robot0', 'message': 'Register', 'listenPort': 65433} from ('127.0.
0.1', 65433)
               Closing connection to 127.0.0.1


############################################################
Added  Robot0  from  ('127.0.0.1', 65433)
There are now  1  robots registered
       Robot Name: Robot0      IPv4: 127.0.0.1
Press Q key after all robots have registered
```

Figure 11: Simulation showing successful connection between the robot and the director on the director screen, emulating a server-client relationship.

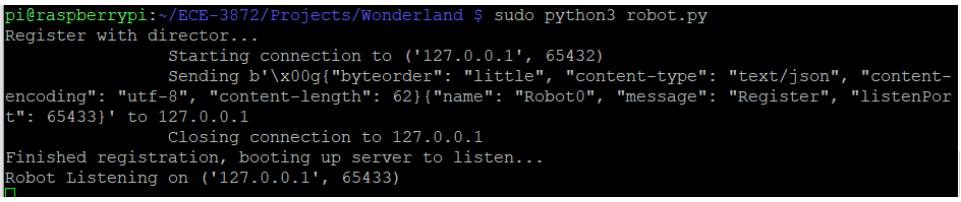


Figure 12: Simulation showing successful connection between the robot and the director on the robot screen, emulating a server-client relationship.

In order to make this connection however, the raspberry pi must be on the same network as the director. This was done through a personal hotspot, and the pi was set up with a WPA supplicant file to automatically connect to the hotspot whenever the hotspot and the pi were on at the same time. Successful execution of the required functionalities (servo movement, script audio, and lights) are shown in the attached video.

Some difficulties we faced were getting the correct distance needed for the servo movement, as it would often over or undershoot how far the robot's horn needed to go. Audio was also a main issue with the robot as we had problems outputting a sound with substantial volume. We later figured out the servo distance through guessing and checking the distance the horn moved, and figured out the audio as a loop within the real-time system (what was under the while True statement) was causing the audio to play in bits and pieces at every real-time loop iteration causing a low volume output that could not have been changed. Removing this loop solved this audio volume issue.

Due to problems with the director, a robot and director connection from the junior design office was not able to be done, so all testing and execution was done locally on the 127.0.0.1 address. This was a class-wide problem.

Schedule

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

Name: Team B06
Project: Master Schedule

	Week Number												
Task	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Brainstorm	Joon												
Team Contract	Erin												
Design Top Level Block Diagram	Woods												
Design Software Block Diagram	Marc												
Proposal													
Title Page Fully Updated		Joon											
Project Summary		Erin											
Block Diagram		Erin											
Schedule		Joon											
Software Diagram		Marc											
Requirements		Woods											
PDR													
Software Design			Erin	Erin	Erin	Erin							
Audio Subsystem with Hardware Design				Erin	Erin	Erin	Erin						
Lighting Function Design				Woods	Woods	Woods	Woods						
Movement Design			Marc	Marc	Marc	Marc	Marc						
Power Subsystem Design				Joon	Joon	Joon	Joon						
Mechanical Design			Joon	Joon	Joon	Joon	Joon						
Interface Design (Combine parts)			Woods	Woods	Woods	Woods	Woods						
CDR													
Hardware Subsystem													
Character Build					Joon	Joon	Joon						
Box build						Joon	Joon						
Hardware/Electrical integration									Woods	Woods			
Movement Subsystem													
Horn Movement build							Joon	Joon	Joon				
Software test with movement subsystem							Marc	Marc	Marc				
Power Subsytem													
Power Subsystem build						Joon	Joon	Joon					
Light Subsystem													
Light subsystem build							Woods						
Software test with light subsystem						Marc	Marc						
Audio Subsystem													
Audio Subsystem build								Woods	Woods				
Software test with audio subsystem					Erin	Erin	Erin	Erin	Erin				
Testing and Debuging													
Software build and debugging					Marc	Marc	Marc	Marc	Marc				
Demonstration test										All	All		
System Integration										All	All		
System Test											All		
Final Inspection and Demonstration													

Milestones	Joon	Electrical/Hardware Design
Tasks	Woods	Electrical/Hardware Design
	Marc	Software Design
	Erin	Software Design
	All	

Table 2 Schedule to complete Project Wonderland

Conclusion

See “End of Life” document.

Appendix A – robot.py

```
import sys
import socket
import selectors
import traceback
import multiprocessing
import time
import keyboard
import csv
import os

from Protocol import libserver
from Protocol import libclient
from gtts import gTTS

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

ROBOT_NAME = "Robot0" #"<INSERT MATCHING ROBOT NAME IN CSV FILE"
HOST = "127.0.0.1" #"<DIRECTOR IP ADDR>"
PORT = 65432 # DIRECTOR LISTENING PORT

LISTEN_PORT = 65433 # ROBOT LISTEN PORT raspberry pi port SSH

# def csv_msgparser(csv_file_path):
#   #robot6_speak = []
#
#   csv_reader = csv.reader(open(csv_file_path)) # open file
#   for i, robot_msgs in enumerate(csv_reader): #essentially the same as i
#     if ROBOT_NAME.lower() in csv_reader: # find robot6's message line(s)
#       robot_msg = [row for idx, row in enumerate(csv_reader) if idx == i]
#       break
#       #multiple lines
#       #robot_msgs = 'robot_msgs[{}] = {}'.format(i, robot_msgs) # create a list
of lists in each index for all messages found for robot6
#   #we only need the third column (the actual line) for this, robot6's messages
found now
#   # for msgs in robot_msgs:
#   #   robot6_speak.append(msgs[2]) #get third column(message) from lines
#
#   return robot_msg

# def msg_to_audio(msg_to_read):
#   language = 'en'
#   speak = gTTS(text=msg_to_read, lang=language, slow=False)
#   speak.save("speak.mp3")
#   os.system("speak.mp3")
```

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

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

  print('Finished registration, booting up server to listen...')

  while True:
    msg = listen_for_director(HOST, LISTEN_PORT, libserver)

    print(msg)
    if msg['value'] == 'break':
      break

    print('Main Loop recieved ', msg, ' so will start to do corresponding task')

    # READ THE MESSAGE AND DO WHATEVER YOUR ROBOT WILL DO.
    os.system('python gpio2.py')
```

Appendix B – gpio2.py

```
import RPi.GPIO as GPIO
import os
import subprocess
from gpiozero import Servo
from time import sleep

#led = 17
#characterled = 23
#button = 22
#servo = 13
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
GPIO.setup(23, GPIO.OUT)
GPIO.setup(22, GPIO.IN, pull_up_down=GPIO.PUD_UP) #led/audio control button

servo = Servo(13)
val = -1
ledState = False
buttonPress = False
os.system("gpio -g mode 12 alt0")

try:
    while True:
        if GPIO.input(22):
            buttonPress = False
            print("FButton")
        else:
            if not buttonPress:
```

```
                buttonPress = True
                ledState = not ledState
                GPIO.output(17, ledState)
                GPIO.output(23, ledState)
                servo.value = val
                sleep(0.1)
                while val < 0.5:
                    val = val + 0.1
                sleep(1)
                servo.value = None

                sleep(1)
                os.system('omxplayer output2.mp3')

                servo.value = .5
                sleep(1)
                while val > -0.5:
                    val = val - 0.1
                sleep(1)
                servo.value = None
        print(val)
except KeyboardInterrupt:
        print("Program stopped")

GPIO.cleanup()
```