# Karaoke Machine Project Manual

## Goals of the Lab

1. Playback microphone input in real-time with manipulations through our Raspberry Pi by utilizing a variety of python modules.
2. Select and modify audio signal manipulations by sending inputs via a web app using the Raspberry Pi as a wireless access point.

## Parts List for the Lab

Raspberry Pi 4b+
USB/XLR Dynamic Microphone
Bookshelf Speakers
IQaudio DigiAMP+
12V 10A Power Supply
Speaker Wire

## Part 1: Install the DigiAMP+ hat

Person Responsible: Ian

1. Ensure no wires or power cords are connected to either the Raspberry Pi or the DigiAMP+.

2. Hand-screw the four 12mm PCB spacers that came with the DigiAMP+ into the corners of the Raspberry Pi.
    a. The spacers are on the top side of the Pi, the screws that hold the spacers are inserted through the bottom of the Pi.

3. Line up the DigiAMP+ with the Pi's 40-pin connector and the 4 new spacers. Press down gently to install.

4. Install the screws that came with the DigiAMP+ into the corners of the DigiAMP+.
    a. Tightening should be less than snug.

## Part 2: Assemble and test Amplifier-Speaker Connection

Person Responsible: Izuka, Ian

1. Connect speaker wire from the terminals on the DigiAMP+ to the terminals on the speakers.
    a. Ensure positive (+) on the DigiAMP+ terminal is paired with the red (+) terminal on the speaker.

b. Ensure negative (-) on the DigiAMP+ terminal is paired with the black (-) terminal on the speaker.
c. Think through how much speaker wire length you may want.

2. You may only apply power to the DigiAMP+ once you are confident that your speakers are connected properly, to avoid damaging the DigiAMP+.
    a. Connect your power supply to the 12V DC barrel port on the DigiAMP+.

3. Connect to your Pi via ssh in Visual Studio Code.
    a. The default password for our Pi is:
        i. Thx2EcsSupp0rt
        ii. The 3rd to last character is zero.

4. Configure the Pi to recognize and enable the DigiAMP+, by modifying the /boot/config.txt in the following ways.
    a. Uncomment the following line by removing the # at the beginning of the line.
        `dtparam=12s=on`
    b. Comment out the following SPI and I2C lines by adding a # at the beginning of the line.
        `#dtparam=12c_arm=on`
        `#dtparam=spi=on`
    c. Comment out the following line by adding a # at the beginning of the line.
        `dtparam=audio=on`
    d. Add the following line.
        `dtoverlay=iqaudio-dacplus,auto_mute_amp`
    e. Comment out any other "`dtoverlay=`" lines, such as "`dtoverlay=vc4-fkms-v3d`"
    f. Comment out the bottom few lines that enable pushbutton shutdown and GPIO pins by adding a # at the beginning of the line.

5. Reboot the Pi to apply the changes.
    `sudo reboot`

6. Check that the Pi recognizes the DigiAMP+ as an audio output using the following command.
    `aplay -l` (note that this is a lowercase L)
        i. You should see the DigAMP+ appear as an output.

7. Start a virtual machine to install modules using the following commands.
    `sudo apt install python3-venv`
        i. This is likely already the newest version and will just inform you that it does not need to be updated.

```
cd /home/pi/Documents
```

ii. IMPORTANT: You do NOT want to run the next command inside of your GitHub directory. This would add thousands of files that are frequently changing to your GitHub repository.

```
python3 -m venv VirtualMachine --system-site-packages
```

iii. The "name" argument is the name of the virtual machine you are creating. You can name it whatever you want.

```
source VirtualMachine/bin/activate
```

iv. This starts the virtual machine.

v. If you want to exit the virtual machine at any point, simply run the "deactivate" command and the virtual machine will shut down.

8. Install the following modules.

```
sudo apt-get update
sudo apt install python3-numpy portaudio19-dev libportaudio2
libasound-dev
sudo apt-get install libportmidi-dev libsndfile1-dev
libasound2-dev portaudio19-dev liblo-dev
pip install pyo
pip install pyalsaaudio
pip install pydub
pip install numpy
pip install sounddevice
```

9. Run the following command to verify that audio is played through both speakers, one at a time.

```
speaker-test -t sine -f 1000 -c 2
```

## Part 3: Integrate a Microphone

Person Responsible: Andrew

1. Connect the microphone's USB cable to the top right USB-A port of the Raspberry Pi.

2. Check that the microphone is recognized as an audio input using the following command:

```
arecord -l
```

(note that this is a lowercase L)

i. You should see your USB microphone appear in the output.

3. In a new directory, create two new files for microphone capabilities called "microphoneTesting.py" and "microphoneInterface.py" using the following commands.

a. Create and enter a new directory called "microphone_test" using the following commands.

```
mkdir /home/pi/Documents/microphone_test
cd /home/pi/Documents/microphone_test
```

b. Create a new file called microphoneTesting.py using the following command.

```
sudo nano
/home/pi/Documents/microphone_test/microphoneTesting.p
y
```

a. Paste the following script into the new file you just created and save it using ^S & ^X.

```
import sounddevice as sd
import numpy as np

print(sd.query_devices())

duration = 5  # seconds
samplerate = 44100

input_device = 1    # Replace with the index of your
microphone (e.g., Fifine Microphone)
output_device = 0   # Replace with the index of your
headphones

print("Recording...")
audio = sd.rec(
    int(duration * samplerate),
    samplerate=samplerate,
    channels=1,
    dtype='float32',
    device=(input_device, output_device)  # specify
both input and output
)
sd.wait()

print("Playing back...")
sd.play(audio, samplerate, device=output_device)
sd.wait()
```

4. Run microphoneTesting.py using the following commands. After speaking into the microphone for five seconds, you should hear the audio you recorded play back through the speakers. Note that this is not a real-time audio system; it is playing back a recording.

a. If you do not hear anything, check the following.
    i. Ensure the DigiAMP+ is not on mute.
    ii. Ensure any settings on the speaker boxes are permitting audio.

```
cd /home/pi/Documents/microphone_test
python microphoneTesting.py
```

5. Now, it's time to create a real-time audio system.
    a. Create a new file called microphoneInterface.py using the following command.

```
sudo nano
/home/pi/Documents/microphone_test/microphoneInterface
.py
```

    b. Paste the following script into the new file you just created and save it using ^S & ^X.

```
from pyo import *
import time
import threading

# Print devices
print("Available devices:")
pa_list_devices()

# Device indices
INPUT_DEVICE = 1
OUTPUT_DEVICE = 0

# Create and configure server
s = Server(sr=44100, buffersize=512, nchnls=1,
audio='portaudio')
s.setInputDevice(INPUT_DEVICE)
s.setOutputDevice(OUTPUT_DEVICE)

try:
    s.boot()
except Exception as e:
    print("Failed to boot server. Check device
indices.")
    raise

s.start()
```

```
# Mic input and pitch shifting
try:
    mic = Input(chnl=0)
    pitch = Harmonizer(mic, transpo=0, feedback=.7)
    pitch.out()
except Exception as e:
    print("Error setting up audio objects.")
    s.stop()
    raise

# Function to listen for pitch commands in another
thread
def pitch_control():
    print("\nPitch control ready. Type a number (e.g.
4, -3) and press ENTER to change pitch in semitones.")
    print("Type 'q' to quit.")
    while True:
        cmd = input("Transpo: ").strip()
        if cmd.lower() == 'q':
            print("Exiting pitch control...")
            break
        try:
            new_transpo = int(cmd)
            pitch.setTranspo(new_transpo)
            print(f"Pitch shifted to {new_transpo:+d}
semitone(s).")
        except ValueError:
            print("Invalid input. Enter an integer or
'q' to quit.")

# Start pitch control thread
control_thread =
threading.Thread(target=pitch_control)
control_thread.daemon = True
control_thread.start()

# Main loop
print("System live. Adjust pitch using the terminal.")
try:
```

```
            while control_thread.is_alive():
                time.sleep(0.1)
        except KeyboardInterrupt:
            print("\nStopping audio.")
            s.stop()
```

6. You should now have a real-time audio system!

## Part 4: Configuring the Pi as a soft access point

Person Responsible: Ian

1. Install hostapd & dnsmasq to handle all of the networking.

```
sudo apt-get install hostapd dnsmasq
```

   a. Verify installation by checking that the following commands return valid paths.

```
which hostapd
which dnsmasq
```

2. Before making changes, save the original "dnsmasq.conf" file by copying it to "dnsmasq.conf.orig" with the following command.

```
sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig
```

3. Configure dnsmasq to establish a range of IPs that the Pi can assign to connected devices.

   a. Open /etc/dnsmasq.conf using the following command.

```
sudo nano /etc/dnsmasq.conf
```

   c. Paste the following lines into /etc/dnsmasq.conf, which you just opened and save it using ^S & ^X.

```
interface=wlan0
dhcp-
range=192.168.4.10,192.168.4.50,255.255.255.0,24h
```

4. Configure hostapd to enable your Pi to become a wireless access point.

   a. Open / etc/hostapd/hostapd.conf using the following command.

```
sudo nano /etc/hostapd/hostapd.conf
```

   d. Paste the following lines into /etc/hostapd/hostapd.conf, which you just opened and save it using ^S & ^X.

```
interface=wlan0
driver=nl80211
```

```
ssid=RaspberryPiWiFi
hw_mode=g
channel=7
auth_algs=1
wmm_enabled=0
macaddr_acl=0
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=PiPassword
wpa_key_mgmt=WPA-PSK
rsn_pairwise=CCMP
```

5. Instruct the Pi where to look for this new information.
    a. Open /etc/default/hostapd using the following command
        ```
        sudo nano /etc/default/hostapd
        ```
    e. Modify the file to include the following line and save it using ^S & ^X.
        ```
        DAEMON_CONF="/etc/hostapd/hostapd.conf”
        ```
    b. Run the following commands to make hostapd executable.
        ```
        sudo systemctl unmask hostapd
        sudo systemctl enable hostapd
        ```

6. Now, create a script that allows you to toggle between “access point” mode and “clinet” mode. In client mode the Pi can access the ECS-IoT Wi-Fi, whereas in access point mode the Pi will be able to host its own Wi-Fi connection to run the web app.
    a. Navigate to ~/Documents using the following command.
        ```
        cd /home/pi/Documents
        ```
    b. Create a file called “toggleWiFi.sh” using the following command.
        ```
        sudo nano /home/pi/Documents/toggleWiFi.sh
        ```
    c. Paste the following script into that new file and save it using ^S & ^X.

        ```
        #!/bin/bash

        MODE=$1

        if [ "$MODE" = "ap" ]; then
            echo "🔁 Switching to Access Point mode..."

            # Stop client services
            sudo systemctl stop wpa_supplicant
            sudo systemctl stop NetworkManager
        ```

```
    # Bring down wlan0 cleanly
    sudo ip link set wlan0 down
    sudo ip addr flush dev wlan0
    sudo ip link set wlan0 up

    # Assign static IP for AP
    sudo ip addr add 192.168.4.1/24 dev wlan0

    # Start AP services
    sudo systemctl start hostapd
    sudo systemctl start dnsmasq

    echo "✅ Access Point mode enabled."

elif [ "$MODE" = "client" ]; then
    echo "🔄 Switching to Wi-Fi Client mode..."

    # Stop AP services
    sudo systemctl stop hostapd
    sudo systemctl stop dnsmasq

    # Reset wlan0
    sudo ip link set wlan0 down
    sudo ip addr flush dev wlan0
    sudo ip link set wlan0 up

    # Start client services (handles DHCP + Wi-Fi
connection)
    #sudo systemctl start dhcpcd
    sudo systemctl start wpa_supplicant
    sudo systemctl start NetworkManager
    echo "✅ Client mode enabled."

else
    echo "Usage: $0 [ap|client]"
    exit 1
fi
```

7. Make your new script executable by running the following command.

```
sudo chmod +x /home/pi/Documents/toggleWiFi.sh
```

8. Run the following command to place your Pi in access point mode.
```
/home/pi/Documents/toggleWiFi.sh ap
```
   a. Run the following command to verify that wlan0 is now available with the expected IP address of 192.168.4.1/24.
```
ip addr show wlan0
```
   b. If that worked, your Pi should now be available in the Wi-Fi settings of any nearby device! It is called RaspberryPiWiFi and its password is PiPassword.

## Part 5: Web App

Person Responsible: Izuka, Ian

1. Run the following command to reconnect the Pi to the ECS-IoT network.
```
/home/pi/Documents/toggleWiFi.sh client
```

2. Ensure you're still inside of the virtual machine by navigating to the Documents directory and running the following command.
```
cd /home/pi/Documents
source VirtualMachine/bin/activate
```

3. Install flask to host the website
```
pip install flask
```
   a. Verify installation by checking that the following command returns a valid path.
```
which flask
```

4. Create and enter a new directory called "flask_web_app" to organize your web app files using the following commands.
```
mkdir /home/pi/Documents/flask_web_app
cd /home/pi/Documents/flask_web_app
```

5. Create a new python file called "web_app.py" to run the web app.
   a. Create web_app.py using the following command.
```
sudo nano /home/pi/Documents/flask_web_app/web_app.py
```
   b. Paste the following script into web_app.py and save it using ^S & ^X.

```
from flask import Flask, render_template, request,
redirect, url_for

app = Flask(__name__)
```

```
counter = 0       # Button counter
volume = 50       # Initial volume (midpoint)

@app.route('/')
def index():
    return render_template('index.html',
count=counter, volume=volume)

@app.route('/increment', methods=['POST'])
def increment():
    global counter
    counter += 1
    print("current counter number =", counter)
    return redirect(url_for('index'))

@app.route('/decrement', methods=['POST'])
def decrement():
    global counter
    counter -= 1
    print("current counter number =", counter)
    return redirect(url_for('index'))

@app.route('/set_volume', methods=['POST'])
def set_volume():
    global volume
    try:
        volume = int(request.form.get('volume', 50))
    except ValueError:
        volume = 50  # fallback
    print("current volume number =", volume)
    return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

6. Create a new “HTMLtemplate” directory inside the flask_web_app directory using the following command.

```
mkdir /home/pi/Documents/flask_web_app/HTMLtemplate
```

7. Create a new HTML file called “index.html” to provide the source code for your web app.
    a. Create index.html using the following command.

```
sudo nano
/home/pi/Documents/flask_web_app/HTMLtemplate/index.h
tml
```

    b. Paste the following lines of HTML code into index.html and save it using ^S & ^X.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>DJ Booth</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/di
st/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body {
            background-color: #121212;
            color: white;
        }
        .slider {
            width: 100%;
        }
    </style>
</head>
<body class="container py-4">
    <h1 class="mb-4">🎧 Team Three's DJ Control
Panel</h1>

    <div class="mb-3">
        <p>Current Count: <strong>{{ count
}}</strong></p>
        <form method="POST" action="{{
url_for('increment') }}">
            <button type="submit" class="btn btn-
primary">Plus 1!</button>
        </form>
        <p> </p>
```

```
        <form method="POST" action="{{
url_for('decrement') }}">
            <button type="submit" class="btn btn-
primary">Minus 1!</button>
        </form>
    </div>

    <div class="mb-4">
        <p>Master Volume: {{ volume }}</p>
        <form method="POST" action="{{
url_for('set_volume') }}">
            <input type="range" class="form-range
slider" name="volume" min="0" max="100" value="{{
volume }}">
            <button class="btn btn-secondary mt-2"
type="submit">Set Volume</button>
        </form>
    </div>

    <hr class="border-light">

    <div class="row">
        <div class="col-md-4">
            <p>Deck A Volume</p>
            <input type="range" class="form-range
slider" min="0" max="100" value="50">
        </div>
        <div class="col-md-4">
            <p>Deck B Volume</p>
            <input type="range" class="form-range
slider" min="0" max="100" value="50">
        </div>
        <div class="col-md-4">
            <p>Crossfader</p>
            <input type="range" class="form-range
slider" min="0" max="100" value="50">
        </div>
    </div>
</body>
</html>
```

8. Run and test web_app.py.
    a. Toggle on the Pi's access point using the following command.
        ```
        /home/pi/Documents/toggleWiFi.sh ap
        ```
    b. Run your web app using the following commands.
        ```
        cd /home/pi/Documents/flask_web_app
        python web_app.py
        ```
    c. The terminal should validate that your web app is running on your Pi.

9. Connect to your Pi's Wi-Fi access point on your phone or laptop. It is called RaspberryPiWiFi and its password is PiPassword.

9. Open a web browser on your device and go to the following address to open your web app.
    ```
    http://192.168.4.1:5000
    ```
        i. Clicking any button should update a value in the Pi's terminal.

10. Create a service that launches your Pi into access point mode automatically on boot.
    a. Create the service by running the following command.
        ```
        sudo nano /etc/systemd/system/togglewifi.service
        ```
    b. Paste the following lines into the new service file and save it using ^S & ^X.

        ```
        [Unit]
        Description=Toggle WiFi Service
        After=network.target

        [Service]
        Type=oneshot
        ExecStartPre=/bin/sleep 10  # Wait for 10 seconds
        before executing
        ExecStart=/bin/bash /home/pi/Documents/toggleWiFi.sh
        ap
        RemainAfterExit=true

        [Install]
        WantedBy=multi-user.target
        ```

    c. Launch the service and ensure that it is running successfully using the following commands.
        ```
        sudo systemctl daemon-reexec
        ```

```
sudo systemctl daemon-reload
sudo systemctl enable togglewifi.service
sudo systemctl start togglewifi.service
sudo systemctl status toggleWiFi.service
```

d. If successful, you should see something like the following.

```
Active: active (exited) since ...
```

## Part 6: Integrate all systems

Person Responsible: Andrew, Izuka, Ian

1. Create a new file called “full_integration.py” and paste THIS code into it.
    a. This code
2. Validate that all the systems are working together as expected

3. Create a service that launches your fully-integrated Python script automatically on boot.
    a. Create the service by running the following command.

```
sudo nano
/etc/systemd/system/full_integration.service
```

    b. Paste the following lines into the new service file.

```
[Unit]
Description=Run Full Integration Script
After=togglewifi.service

[Service]
ExecStartPre=/bin/sleep 10  # Wait for 10 seconds
before executing
ExecStart=/usr/bin/python3
/home/pi/Documents/ELC/Documents/ELC4438-Speech-
Jammer/flask_app/full_integration.py
WorkingDirectory=/home/pi/Documents/ELC/Documents/ELC
4438-Speech-Jammer/flask_app
User=pi
Group=pi
Restart=always

[Install]
WantedBy=multi-user.target
```

c. Launch the service and ensure that it is running successfully by running the following commands.

```
sudo systemctl daemon-reexec
sudo systemctl daemon-reload
sudo systemctl enable full_integration.service
sudo systemctl start full_integration.service
sudo systemctl status full_integration.service
```

d. If successful, you should see something like the following.

```
Active: active (exited) since ...
```

4. Now your Pi should be ready to run all by itself! Reboot your Pi to see if everything launces as intended using the following command.

```
sudo reboot
```