Drum-Glove

With this project, we’re taking something an action as basic as tapping your fingers on the table and turning it into real-time sound. Users would be able to just slide on this glove like any other, and start tapping away as the rhythm of their fingers produces drum beats from a laptop. The Arduino implementation was jointly handled by Aditya, Shreesh, Nirmit and Sanjana.

*Basic idea/aim :*

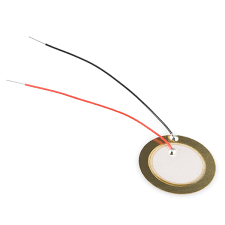
Five piezoelectric sensors represent five fingers on the glove. These are connected to an Arduino Microcontroller. Output is in the form of drum-kit sounds on the speaker of the laptop that the microcontroller is interfaced with. Each finger has a unique sound associated with it, and completely programmable.

*Things needed :*

* An arduino microcontroller



* A bread-board for all connections
* Connecting pins and wires
* Power source
* 5 piezoelectric sensors for each finger on the glove //4 washers with jumper cables soldered onto them



* A glove that fits over your hand

The source code for the input signals was edited and compiled on the Arduino IDE©️ (Windows x64).

The source code and integration for the output was compiled on Processing©️ v3.5.3, and Python v3.6

*The following is a chronological flow that the team followed :*

\*Step 1 : This mainly included brainstorming ideas and flows for our plan, choice of IDE and sensors. We did a brief comparison amongst piezoelectric sensors, pressure sensors, simple switches, and light sensors. Eventually, piezoelectric sensors were chosen because of their ease-of-use and sensitivity.

\*Step 2 : In this next step, one individual sensor was used with the Arduino to test the functioning. We received a binary output on touching a finger to it. This confirmed its use in the tip of the glove, that is, it will trigger a response when tapped on the table. Only the Arduino IDE was used in this.

Step 3 : On the second day, we programmed on the Processing©️ to use the input from the Arduino to produce drum-kit sounds on the laptop’s speaker. We imported .mp3/.wav files to the program to link our code to the output, however, we faced a NullPointerException during execution. This revealed that the program/GPU couldn’t locate the necessary drivers to utilise.

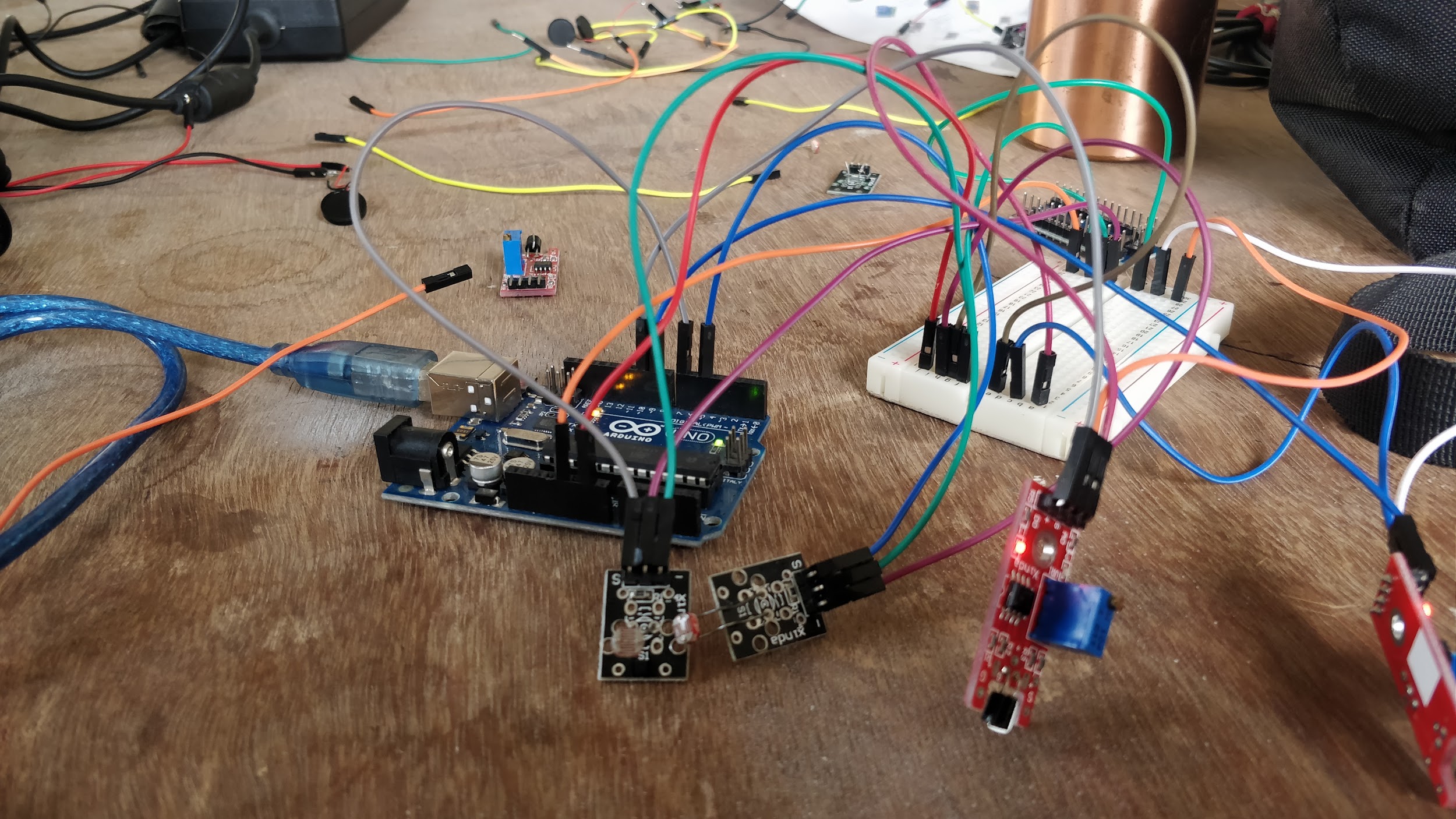
\*Step 4 : Since the software mentioned above had failed to integrate the input and the output, we decided to try another approach. We have now used a Python script to input and decode the serial output of the arduino. Using this decoded input, we find out which sensor was pressed. We played the corresponding file using winsound and playsound, pre-existing python library files. We faced a problem, namely the .wav/.mp3 files were unable to be played with this library due to some issues that we discovered only later. In the end, as a compromise, we decided to prototype the final response using beeps of different frequencies instead of different files, until we were able to finally play the sound files.

\*Step 5 : On day 3, we fixed the problem with being able to play sound files; there was a permission access problem that was easily solved. Once we were able to play the files, we ran the code and this is the response we got from the arduino: <https://drive.google.com/file/d/12KEr5KNPQkMVsYtZVcyK__uiuRSLPnnq/view?usp=sharing>

\*Step 6: Then we soldered the piezoelectric sensors and tried running the code again. When the sensors weren’t triggering the sounds we realized that the specifications of the required piezo didn’t match with the one which we were using. So we ordered new ones.

\*Step 7: We discovered that the piezo-electric sensors we were using were rated at 9V, not according to our specifications, and we needed sub-5 V sensors, and hence, they malfunctioned. We decided to try out touch sensors and light detecting sensors (LDRs). Unfortunately, we found that LDRs are susceptible to changes in the surrounding temperature, and the readings we recorded were inaccurate. Eventually, we settled on touch-sensors and the following is a video of the penultimate step.\

(insert LDR video drive link)



Step 8 : Finally, in order to simplify the working of the prototype, we decided to use switches. Only 4 fingers can act as triggers; we press the fingers to a grounded terminal that is attached to the thumb. So to play sounds, we touch the fingers to the thumb. This idea made it to the final *working* model. The triggers, the arduino and the wires were attached to a glove and we were able to wear it and play a few beats on the speaker.

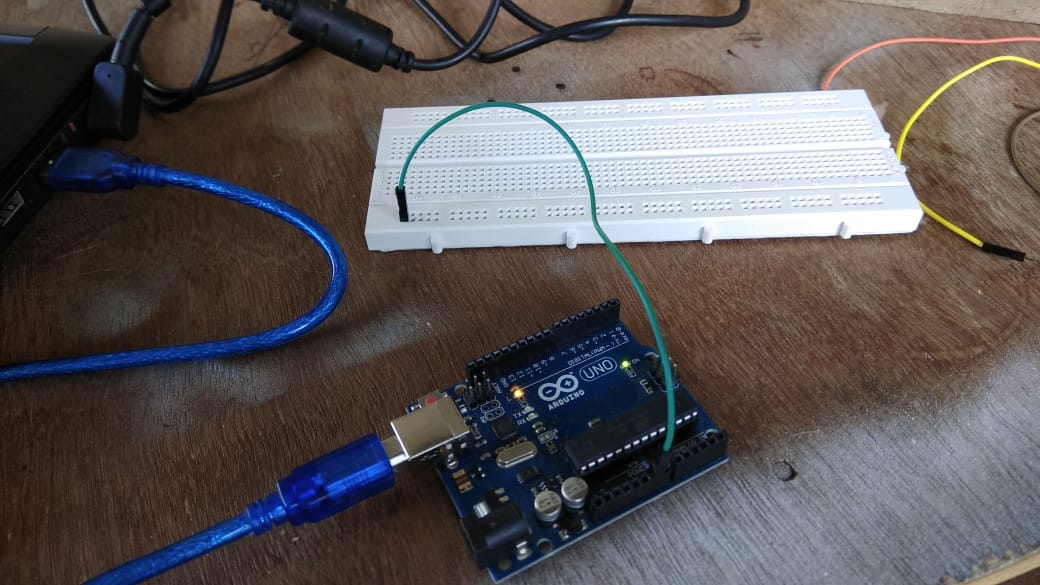
The steps that are starred are the steps that need to be taught to the ones who are taking the course. Apart from this, basics about arduino coding and python need to be taught on the first day itself. Interfacing between the arduino and the PC also have to be told to the students.

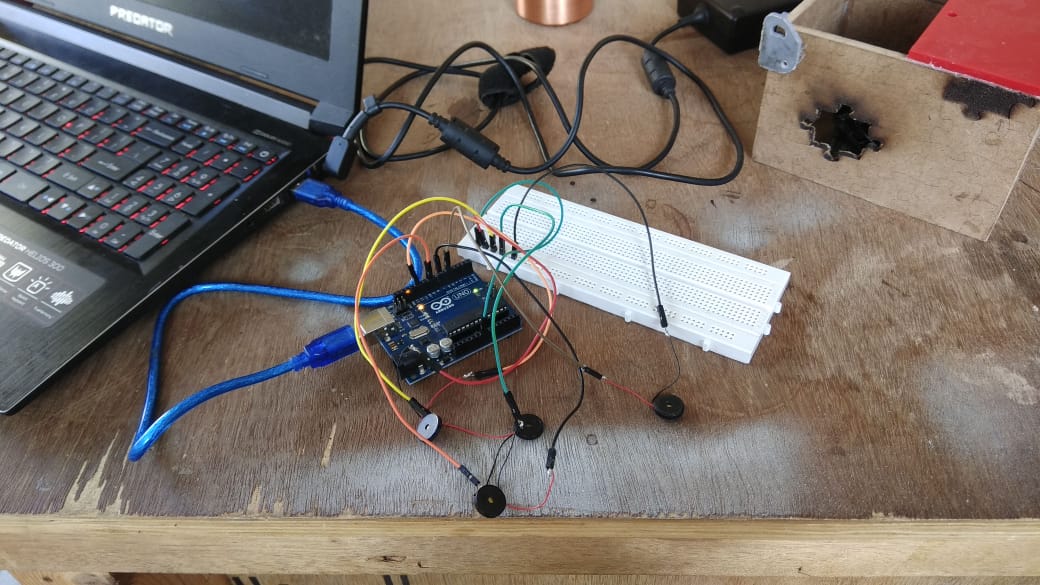
*What was learnt (by us):*

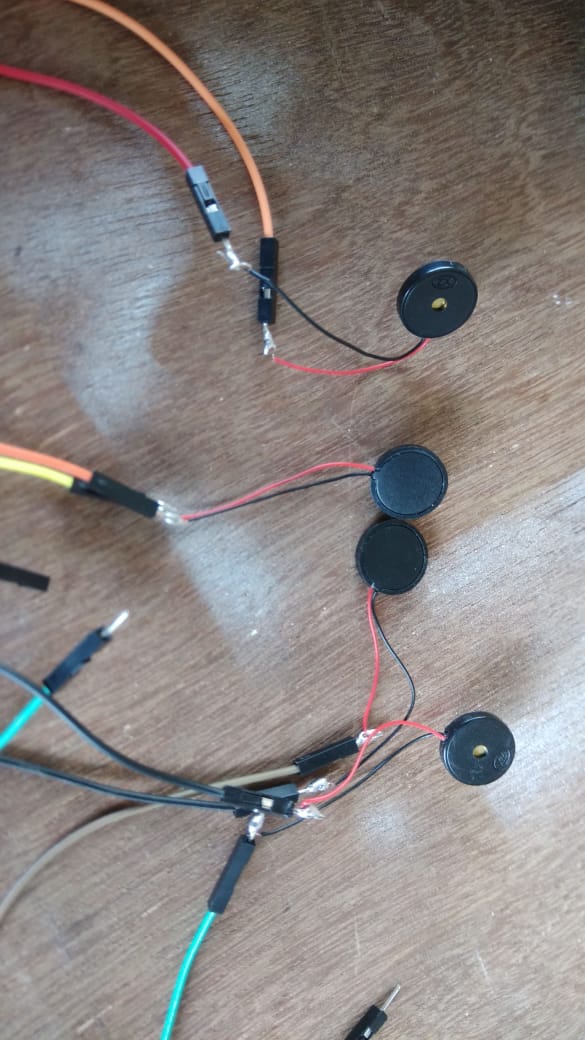
When we started with the project we thought of the workflow as being very straightforward. The components were simple: an arduino, a few sensors and a laptop. The glove was only required at the last stage, not to make the prototype. Once we actually started researching and working on the code, we realized that triggering actions on the laptop using an arduino was our main challenge. We learnt how to interface between an arduino and a laptop using python scripts. We learnt to work with and solved unexpected coding challenges that appear on the go. We also realized that the specifications of certain components are very important to the working of the project. Finally, now we can say and be sure that if any other project involves this kind of interfacing, we can make it happen quickly and without encountering similar problems.

The drive folder that includes all the pictures and videos of the project; this includes the video showing how the piezo sensors do not work: <https://drive.google.com/open?id=1zqlqY0jhtfSM3CoZ1u1Vn25_CLm_HuuU>

* Arduino connected to the breadboard:



* Sensors connected to the breadboard
* Piezo sensors:



**Raspberry Pi Implementation**

Once we were done with making the glove using an Arduino, we wanted the glove to be more of a standalone piece of equipment, only requiring a power source and a speaker. For this, we used a Raspberry Pi, a sort of computer on its own. This part was handled only by Shreesh.

The full working model was made in 3 days. The steps followed were:

**Step 1 (Setup):** On day 1, I received an RPi. I prepared an SD card for the storage requirements (formatting etc.) and downloaded the Raspbian OS in the form of the NOOBS software. Once I transferred it to the SD card, I inserted it into the RPi and installed the OS. Downloading and installation took the most amount of time.

**Step 2 (coding):** Once the setup was done, on the second day I wrote a python script to use the GPIO pins as triggers. In the script, I take input from separate pins to figure out which music file to play. This script also uses the pygame module (pre-installed as part of the python library) to play sounds. For initial testing purposes I chose to print output rather than play music.

**Step 3 (testing):** I had to figure out how the GPIO pins are triggered. I wrote a separate script to figure out the behavior of each pin, as I was encountering garbage values. After testing, I found out that only 2 pins sensed a High input by default, while the rest sensed garbage values by default. With this began the quest to get rid of these garbage values and try to force the pins to sense only one value by default.

**Step 4:** On the third day, I first tried a way to “pull down” the voltage value read by the input pin. When this didn’t work, I tried to create a sort of switch mechanism to force high or low values as input. This would end up shorting the 3.3V and ground pins of the Pi, so the idea of switches was rejected. Next was the idea of assigning 4 pins 4 different bit streams of 4 bits (0000, 0001, 0011, 0111) which the input pin could read and decode (based on number of 1’s in a group of 4 bits). The 4 different pins could be assigned to the 4 different sounds. The pin that reads 1 as default could be used to differentiate the 4 inputs. Implementing this idea would require using multiple threads to output the bit streams continuously on 4 pins, a highly impractical idea on an RPi. Hence, I went back to a simpler method.

**Step 5:** I went back to simply reading from 4 GPIO pins, but I used one pin as a permanent HIGH output. Almost miraculously, this solved the garbage value problem. I compiled a few drum samples that worked with the pygame module. Once the testing was done, I replaced the arduino on the glove with the RPi, made the same connections to the sensors, and the working model was complete.

**Step 6:** To pull it all together, I modified the .bashrc file of the RPi to run the python script whenever the RPi is switched on or when a terminal is opened. Keyboard shortcuts can be used to open/close the terminal and to close the python program. After doing this, only a speaker and a power cord is required to run the drum glove. I modified the code a bit so that the delay between a finger touch and the sound playback is reduced, and one finger touch only plays the sound once.

**Problems faced:**

A different platform meant a different set of problems. As the sensor issues were fixed, I didn’t have to focus on that much. The main issue was the garbage values being read by the RPi pins. The final issue was loose connections of the jumper cables (as you can see in the video ahead, I had to hold the connections in place with my other hand).

**What was learnt:**

I learnt how to deal with binary output of the pins. A lot of brainstorming was involved with regards to getting around the GPIO pins’ behavior. I became comfortable with the RPi environment, as well as python’s pygame module.

**How to teach:**

The RPi implementation serves as a logical step forward from the arduino implementation. This time around, there is no interfacing between the glove and the PC; the glove itself is the PC. So one of the major steps in the arduino implementation is missing in the RPi implementation.   
  
The students, by this time, must have become comfortable with dealing with python scripts and using the pins as input/output. Hence, this time we should focus on making them comfortable with the Linux environment, as the Raspbian OS is based on Linux.

Using the Linux terminal to navigate files and to run programs/scripts should be taught right after setting up the RPi. The python script used by me doesn’t properly run on the IDLE terminal, so running it on bash terminal is essential (hence the focus on the bash terminal).

It’s recommended for the students to come up with their own ideas to implement the logic for triggering sounds with the fingers. They may also encounter the garbage value problem with the GPIO pins, and if they get stuck on it for too long, they’ll need to be told of the trick that I used. If they can come up with a better solution, so be it.

The 16 bit .wav sound files can be provided directly to the students, or they may search for them online and download them.

At the end of the day, the students will learn how to deal with input and output on an RPi and how to work on linux terminal.

The photos and videos:

<https://drive.google.com/drive/folders/1ZdPZ8IR9oV2ysrImjGFZpxC7HbRai4Q9?usp=sharing>

**References:** <https://makezine.com/projects/make-33/simple-soundboard/>

<https://www.raspberrypi-spy.co.uk/wp-content/uploads/2014/07/Raspberry-Pi-GPIO-Layout-Model-B-Plus.png>