Technical Report: Smart Speaker

Contributing Team Members: Kevin Huang, Po-Jen Ko, Eric Lee, David Thomas

ECE Honors Lab SP-22

29 April 2022

Introduction

Our project is to create a smart speaker that is aware of its surroundings. What this means is that our smart speaker would be able to detect a human in the room and play music to the appropriate volume depending on the relative location from the person to the speaker. This speaker enables a better listening experience as it allows the listener to fully engage with the music at the appropriate volume. In addition, the speaker will be able to turn itself off when it detects nobody in the room.

Purpose

With the global pandemic going on, people are often spending more time inside and by themselves. Being alone can sometimes make a person go crazy, and I know from experience that when I am by myself, I tend to move around a lot. With the speaker being able to constantly adjust to what I am doing, it would create an environment for me to fully enjoy whatever activity I am doing, while listening to great music. Our initial project design was to create a smart speaker that would be able to turn on or off, self-adjust its volume depending on the relative distance from the person to the sensor, and if the speaker picked up any human noise (talking, etc.)

Design

Our device consists of five main components as you can see in Figure 1: the Arduino board, the PIR motion sensors, the ultrasonic sensors, sound sensor, and a speaker. The Arduino board implements the code we want the device to perform. As for the PIR sensor and the Ultrasonic sensor, they detect the user. If the PIR motion sensor detects any motion from the user in a certain direction, it will make the device play music and adjust the volume output according to the distance of the user measured from the Ultrasonic sensor in the same direction that the user was detected. The further the user is, the louder the music. If both PIR sensors detect motions from different directions, then the program will make the adjustment to the volume by using the distance that is further in order to let the person at a further place hear the music. When the PIR sensors do not detect any motion from the user, it will turn off the music automatically because there is no one around. In addition, the sound sensor will also detect ambient sound in the room to ensure the music is loud enough for the user to hear. The entire process is shown in Figure 2. Not to mention, since we didn't get the sound sensor on time, we decided to not use it.

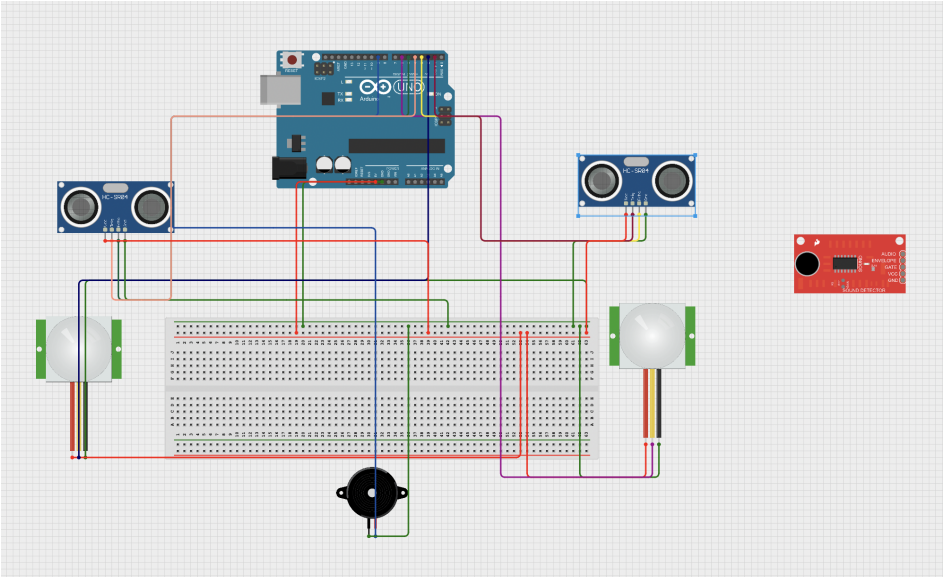

Figure 1. Schematic of Our Design

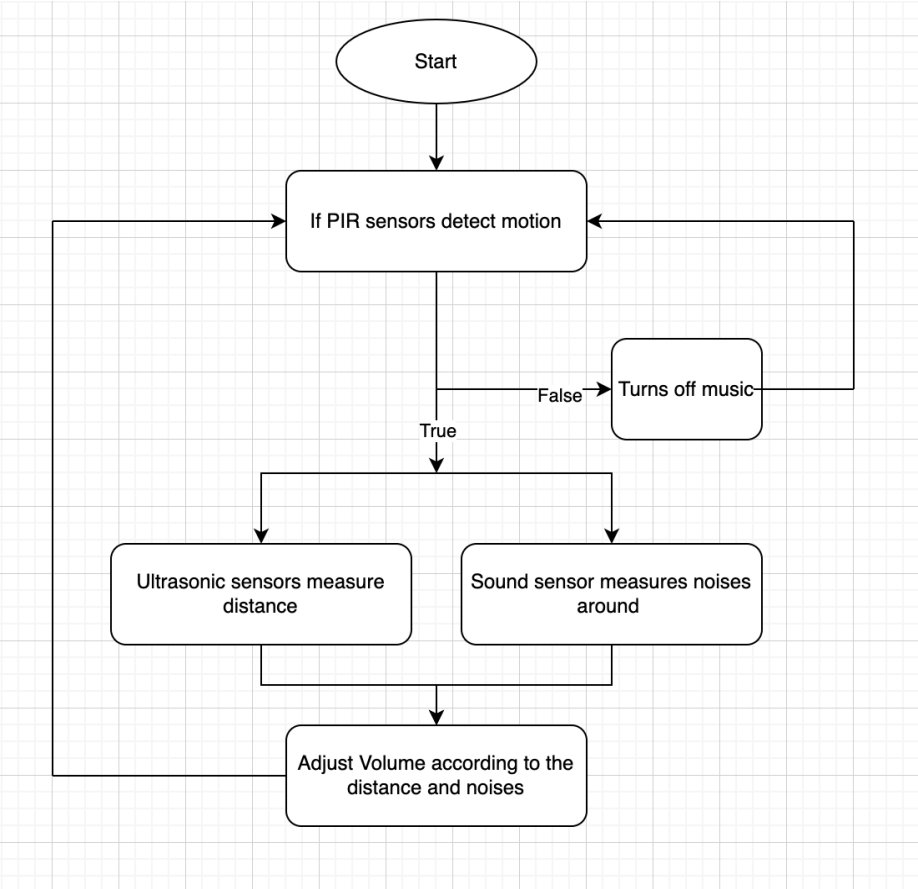

Figure 2. Flow Chart of the Smart Speaker

For the code of our project, we use the standard coding format for ultrasonic sensors and PIR sensors which sets up the pins for sending out and receiving signals and the pins for reading the value from PIR sensors. In addition, for the speaker, the program uses the library for pitches and defines the notes and duration as shown in Figure 3.

ECE120_Honors_Lab | pitches.h

```
// First direction set up:

int pirPin1 = 2;                // PIR Out pin
int pirStat1 = 0;               // PIR status
int trigPin1 = 1;
int echoPin1 = 3;
long duration1; // variable for the duration of sound wave travel
int distance1; // variable for the distance measurement

// Second direction set up:

int pirPin2 = 6;                // PIR Out pin
int pirStat2 = 0;               // PIR status
int trigPin2 = 4;
int echoPin2 = 5;
long duration2; // variable for the duration of sound wave travel
int distance2; // variable for the distance measurement

#include "pitches.h"

// notes in the melody:
int melody[] = {

  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
};

int buzzer = 9;
int volume = 0;

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {

  4, 8, 8, 4, 4, 4, 4, 4
};



void setup() {
  pinMode(pirPin1, INPUT);
  pinMode(trigPin1, OUTPUT); // Sets the trigPin as an OUTPUT
  pinMode(echoPin1, INPUT); // Sets the echoPin as an INPUT

  pinMode(pirPin2, INPUT);
  pinMode(trigPin2, OUTPUT); // Sets the trigPin as an OUTPUT
  pinMode(echoPin2, INPUT); // Sets the echoPin as an INPUT

  Serial.begin(9600); // // Serial Communication is starting with 9600 of baudrate speed
}
```

Figure 3. Setup Code

In the main code for ultrasonic sensors, the code will have trigger pins sending out signals for 10 microseconds and have echo pins to receive the signal. Then the program calculates the distance by using the formula $Distance = Velocity * Time$, where the velocity will equal the speed of sound and time will be half of the travel time of the signal as the signal runs back and forth as shown in Figure 4.

```
void loop(){
  // Clears the trigPin condition
  digitalWrite(trigPin1, LOW);
  delayMicroseconds(2);
  // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
  digitalWrite(trigPin1, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin1, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration1 = pulseIn(echoPin1, HIGH);
  // Calculating the distance
  distance1 = duration1 * 0.034 / 2; // Speed of sound wave divided by 2 (go and back)

  // Displays the distance on the Serial Monitor

  Serial.print("Distance1: ");
  Serial.print(distance1);
  Serial.println(" cm");

  digitalWrite(trigPin2, LOW);
  delayMicroseconds(2);
  // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
  digitalWrite(trigPin2, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin2, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration2 = pulseIn(echoPin2, HIGH);
  // Calculating the distance
  distance2 = duration2 * 0.034 / 2; // Speed of sound wave divided by 2 (go and back)

  Serial.print("Distance2: ");
  Serial.print(distance2);
  Serial.println(" cm");
```

Figure 4. Code for Ultrasonic Sensor

For PIR sensors, the code simply reads the value from the sensors as it only returns a value of 1 when there's motion detected and 0 when there's no motion detected as shown in Figure 5.

```
pirStat1 = digitalRead(pirPin1);
pirStat2 = digitalRead(pirPin2);

Serial.print("PIR1: ");
Serial.println(pirStat1);

Serial.print("PIR2: ");
Serial.println(pirStat2);
```

Figure 5. Code for PIR Sensor

After having both the distances to the user and the values from the PIR sensors, the program sets up logic conditions for adjusting the volume as shown in Figure 6. If no motion is detected, the volume will be 0. Or if both of the PIR sensors detect motions, then the program will take the distance that is further so the user at the further distance can hear the music. Otherwise, if only one of the PIR sensors detect motion, then the volume will be adjusted only based on the distance that is measured in the same direction as the PIR sensor.

```
if(pirStat1 ==0 && pirStat2 ==0){
  volume = 0;
}
else if(pirStat1 == 1 && pirStat2 ==1){
  if(distance1<distance2){
    volume = distance2;
  }
  else{
    volume = distance1;
  }
}
else if(pirStat1 == 1){
  volume = distance1;
}
else{
  volume = distance2;
}

Serial.print("Volume: ");


Serial.println(volume);


}
```

Figure 6. Logic Conditions for Adjusting the Volume

Last but not least, the program will adjust the volume of the buzzer based on the value from the previous code. Then, the program will play the music set up as shown in Figure 7.

```
analogWrite(buzzer, volume);

for (int thisNote = 0; thisNote < 8; thisNote++) {
  int noteDuration = 1000 / noteDurations[thisNote];
  tone(buzzer, melody[thisNote], noteDuration);
  int pauseBetweenNotes = noteDuration * 1.30;
  delay(pauseBetweenNotes);
  noTone(8);
}
```

Figure 7. Code for Playing Music

Challenges and Difficulties

During the process, we encountered a few challenges and difficulties that made our result slightly different from our initial expectations. First of all, we encountered the issue of forming a team at the beginning of the project. We had started off with two members, looking for a third.

However, when we were first ready to start working on a project, one of the additional teammates left the team due to their lack of interest in the project. We then decided to explain the project to our potential new teammates before they joined, and were able to finally form our team after 3 weeks. While this setback might seem minor, it was incredibly frustrating to have a total of three people bail on us because either our project idea was not what they were interested in, or they could not fit the time requirements. It made us lose some confidence in our idea and we had thought about scrapping it altogether.

Another major issue we experienced was part ordering. When we ordered our parts in the first round of ordering, the Raspberry Pi was out of stock. This was challenging since the Raspberry Pi was the most important piece of our project. Since we could not obtain the Raspberry Pi, we decided to use an Arduino Uno instead. A month after we ordered our parts, we only got the PIR sensors. Not only that, we were unable to acquire our sound sensors. Therefore, we had to sacrifice some functions of our speaker due to the lack of components. Despite these supply issues, we decided to work on the coding portion first to maximize efficiency. Fortunately, at the beginning of April, we acquired our Arduino board and finished our code. We then built up our smart speaker in two weeks and started to debug the code.

During the process of debugging, we met another difficulty. Unlike Raspberry Pi that we originally planned to use for our project, the Arduino Uno cannot multitask. For example, it cannot play music and measure the user's distance simultaneously. Instead, it can only sequentially measure the user's distance, adjust the volume, and play the music. This caused some delay when playing the music because while the sensors only took a few milliseconds to complete, it was still enough time to throw off the music.

Results

In the end we were able to accomplish some of the goals we set at the beginning of the project. We successfully created a machine that could detect the motion of an object and measure the distance between the person and the machine (cm). Additionally, the speaker could play rhythmic music. Due to the limitations of the Arduino board, the code could not run both the music and the tests (motion and distance) at the same time. Our original design with the Raspberry Pi would have worked better because that system can run multiple codes at the same time, but unfortunately all the Raspberry Pis have been out of stock for months. Additionally, we never recieved our sound sensor so we were unable to add that to our project.

Future Plans

In the future, we plan to replace the Arduino Uno board with the Raspberry Pi so our code can maneuver the sensors and speaker simultaneously. This will effectively solve our initial challenges with the speaker's capabilities. Currently, our speaker only has two ultrasonic sensors and PIR sensors in two different directions. In order to improve the functionality and accuracy of our smart speaker, we plan to increase the number of sensors to allow a 360˚ view of the surroundings. If the budget is higher, then we can even implement an infrared camera to detect the users more accurately.

References

[1]“Ultrasonic Sensor HC-SR04 with Arduino Tutorial.” *Arduino Project Hub*, https://create.arduino.cc/projecthub/abdularbi17/ultrasonic-sensor-hc-sr04-with-arduino-tutorial-327ff6.

[2] OBJECT AND FINAL VISUALS – Physical computing. “KY038 Microphone Module and Arduino Example.” *Arduino Learning*, 17 July 2018, http://arduinolearning.com/code/ky038-microphone-module-and-arduino-example.php.

[3]“PIR Motion Sensor: How to Use PIRS W/ Arduino & Raspberry Pi.” *Arduino Project Hub*, https://create.arduino.cc/projecthub/electropeak/pir-motion-sensor-how-to-use-pirs-w-arduino-raspberry-pi-18d7fa.

[4]“MAKING A RASPBERRY PI SMART SPEAKER: Using MoodeAudio.” *YouTube*, YouTube, https://www.youtube.com/?gl=DE.