Cal Poly Pomona

The Ultrasonic Theremin

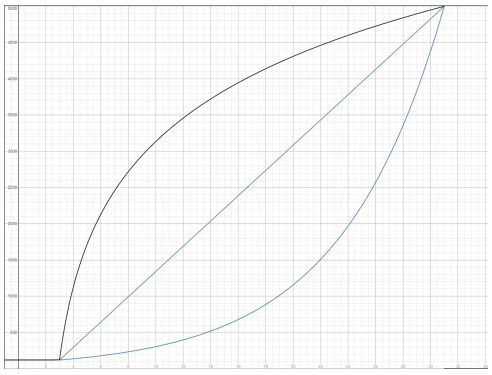
Kyler Martinez
MU3100 Section 5
Professor Sarah Wallin-Huff
May 15th, 2021

Part I: The Beginning

The Ultrasonic Theremin (UST) is inspired by the theremin and was originally designed to function similarly to one by having one hand control the pitch and the other the volume. I was initially inspired when I saw a YouTube video of an individual creating a MIDI theremin but using sensors, and that gave me the idea that I would be able to accomplish that.

The original concept included two modes of operation, one for playing two speakers, each with a different pitch but constant volume, and the other with each speaker playing the same tone but having a volume range. I purchased ultrasonic range sensors, and in each configuration, one sensor would control the left pitch/pitch and the other right pitch/volume. The design also would have two buttons to change the function for calculating the frequency for each speaker and foot pedals that would hold the frequency for each speaker when pressed. While playing with a potentiometer, I figured that would be the best way to change the volume, but the real trouble came when trying to use it.

I found issues working with the potentiometers, more on that in Mark II, but I came into this project knowing far too well I would encounter issues while building the instrument. I expected to create the UST's two-speaker functionality, but I was unsure of the dynamic volume changes. I also expected to gain experience both in the music field and my major in Computer Engineering.



$$F_1(d) = \left\{ d \le d_0 : f_0,\ d_1 \ge d \ge d_0 : \text{floor}\left(f_0 e^{\frac{(d-d_0)}{\frac{d_1-d_0}{\ln\left(\frac{f_1}{f_0}\right)}}} \right),\ d > d_1 : 0 \right\}$$

$$F_2(d) = \left\{ d \le d_0 : f_0,\ d_1 \ge d \ge d_0 : \text{floor}\left(\frac{f_1 - f_0}{d_1 - d_0}(d - d_0) + f_0 \right),\ d > d_1 : 0 \right\}$$

$$F_3(d) = \left\{ d \le d_0 : f_0,\ d_1 \ge d \ge d_0 : \text{floor}\left(\frac{f_1 - f_0}{\log(d_1 - d_0 + 1)} \log(d - d_0 + 1) + f_0 \right),\ d > d_1 : 0 \right\}$$

$$f_0 = 120$$

$$f_1 = 5000$$

$$d_0 = 3$$

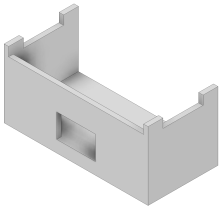
$$d_1 = 31$$

Part II: Early Designs & Problems

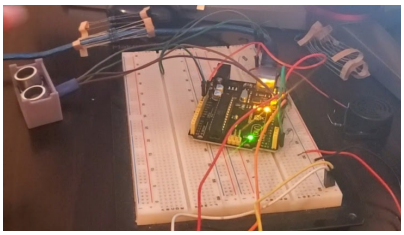
My first act of business was to figure out how the sensors worked and then create a mathematical model relating the distance from the sensor to the frequency of the tone. I developed three different ones, where F is the frequency, d is the distance, f0 and f1 are the min and max frequencies, and d0 and d1 are the min and max distances.

The first problem I encountered was finding an Arduino library that would allow me to alter the tone of the speaker and the volume without extra hardware and output a tone to two speakers

simultaneously. I never managed to find a library like that, so I prioritized a two-speaker design.



To house the sensors, I 3D printed a small chassis for them. The design would not be used in the Mark II but return for the final design. My Mark I for the UST only tested the sensors and the functions to output a tone to each speaker. The Mark I was made using a breadboard, but later designs will use perf boards instead. The Mark I did not have any problems in function, but at this stage, the code and electronics were simple.



To record distance with the sensors, the trigger pin was set to high, and the echo pin was used to find the time it took to return and use that to calculate the time. The code for the sensor [1] was developed by Arbi Abdul Jabbaar, and I used it to create a function that I could reuse for each sensor.

```
double getDistance(int tpin, int epin)
    {
    digitalWrite(tpin, LOW);
    delayMicroseconds(2);
    // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
    digitalWrite(tpin, HIGH);
    delayMicroseconds(10);
    digitalWrite(tpin, LOW);
    duration = pulseIn(epin, HIGH);
    return duration * 0.034 / 2;
    }
```

In the *getData* function, I use a switch case that will activate the function to calculate the frequency for the tone depending on the operation mode. I have included one of the frequency calculation functions, but they follow the same format except for the calculation formula. All the frequency calculation functions are present in the full display of the code. Whenever an object was within 2cm of the device, the read distance would be greater than a thousand or negative. To account for this, I used a max and min distance and returned a frequency of 0Hz if the measured distance was not in the space between the max and min distances.

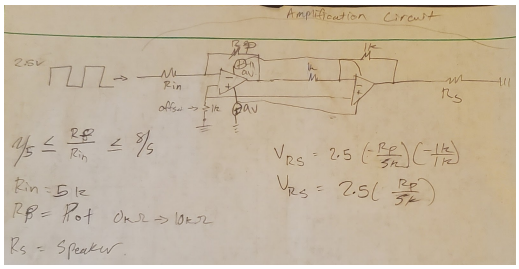
```
int getData(int mode, double d,double dmax, double qmin, double qmax)
    {
     if (d>dmax ||d <3) //If an object is not close enough or too close, it will not play the speaker.
        {
          return 0;
        }
    switch (mode)
    {
      case 0:
        return abs(linearData(d,dmax, qmin, qmax));
        break;

      case 1:
        return abs(logData(d,dmax, qmin, qmax));
        break;

      case 2:
        return abs(expData(d,dmax, qmin, qmax));
        break;

      default:
        return 0;
      }
    }

int linearData(double d,double dmax, double qmin, double qmax)
    {
      return ((qmax-qmin)/(dmax-3)) * (d-3)+qmin;
    }
     void play_tone(Tone x, int f)
     {
       return f<=0 ? x.stop() : x.play(f);
     }
```

Finally to play tones, the Tone library[2] was used to output simultaneously with two speakers. I

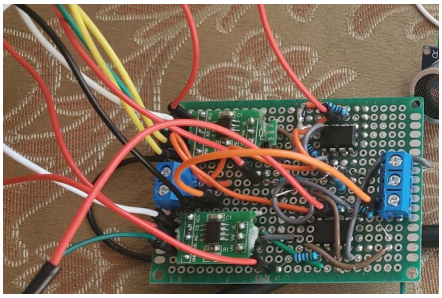
[1] Arbi Abdul Jabbaar, Ultrasonic Sensor HC-SR04 with Arduino Code for Ranging Test
[2] Brett Hagman & Fotis Papadopoulos, Tone Library

created a function built upon the methods provided with the Tone class to play a tone. The play method was used if the frequency did not equal 0Hz but would use the stop method if frequency equaled 0Hz since the speaker would make noise if *play()* was used at 0Hz.

Part III: The Volume Problem

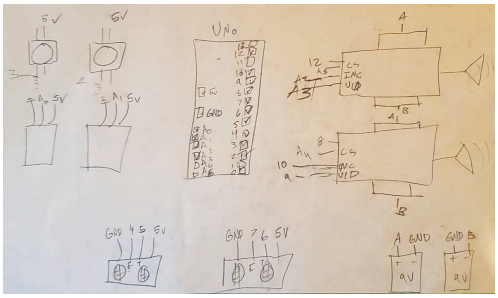


Initially, I wanted to use a digital potentiometer in series with the speaker, but the X9C103S potentiometer had a current limit that was too low for this solution. My next solution was to create an amplifier circuit. I would have used a non-inverting amplifier, but that circuit configuration only allows for a gain greater than or equal to one, which would not work for me since I wanted some volumes to result in a little gain. However, the speakers could not work with a reversed polarity, so I added another amplifier to invert the polarity back to positive voltage. I intended to use an LM358 dual op-amp for each speaker. For the rail voltages, I would use two 9 volt batteries.



I created a subcircuit for the entire amplifier circuit on a perf board and spent a large amount of time soldering the components together. The subcircuit had inputs for the potentiometers, tone outputs from the Arduino, and an output from the op-amp to go to the speaker board, discussed in part IV.

Part IV: The Mark II



The Mark II was my first attempt at creating the overall instrument with all the components operating, ideally, together. The function buttons and foot pedals would be designed as a button with the reading node, pin, at 5V if pressed or at GND if not. The Mark II had an Amp board, speaker board, and two sensor boards. The sensor board contained one sensor, one push button, and a foot pedal for each hand. In my original drawing the buttons were drawn with the wrong pin placement since the nodes were from left to right rather than up and down.

Mode Selection

$\overline{CS}$	$\overline{INC}$	U/$\overline{D}$	MODE
L	↓	H	Wiper Up
L	↓	L	Wiper Down
↑	H	X	Store Wiper Position
H	X	X	Standby Current
↑	L	X	No Store, Return to Standby
↓	L	H	Wiper Up (not recommended)
↓	L	L	Wiper Down (not recommended)

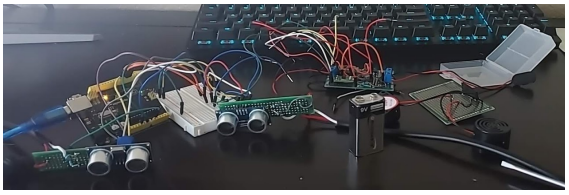
To use the potentiometer, I had to follow the truth table provided with a datasheet[3] online. In the *changeR* function, the *way* variable helps to set the mode of the digital resistor to increase or decrease the resistive output. I also create a square wave to activate the wiper up or down mode described by the INC pin. The *changeVolume* function changes the resistance n times depending on the current and next state of the device so that the program can increase or decrease the resistance enough times.

```
void changeR(int csp,int incp,int upp,bool way)
  {
    digitalWrite(csp, LOW);
    int x = LOW;
    x = way ? HIGH: LOW;
    digitalWrite(upp, x);
    digitalWrite(incp, HIGH);
    delayMicroseconds(5);
    digitalWrite(incp, LOW);
    digitalWrite(upp, LOW);
    return;
    }

void changeVolume(int current, int next, int csp,int incp,int upp)
{
 int n = next - current;
 bool way = n>0 ? true: false;
 n = abs(n);
  for (int i =0; i<n; i++)
  {
    changeR(csp,incp,upp,way);
    }
  }
```

The *OneSpeaker* function handled the program for the mode of operation of each speaker outputting the same tone but with the ability to change the volume. If any of the buttons are read to be pressed, at high, the mode will increase. If the foot pedals are depressed, the frequency and volume will not change. I used the variable *v* to store the old volume before we overwrite it with the calculated volume in the *ChangeVolume* function. Finally, the *play_tone* function will be called for each speaker to play the tone. I will discuss the two speaker code in the Mark III section, but an earlier implementation is in the code collection for Mark II.

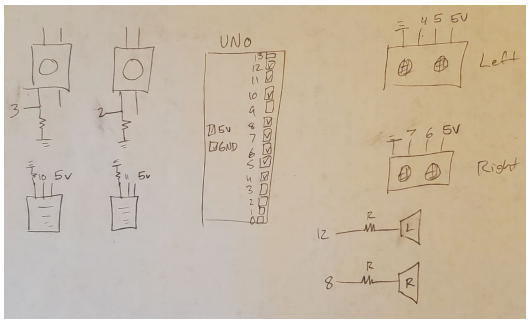
```
    void OneSpeaker()
        {
        if(digitalRead(LBUT == HIGH))
        {
          Lmode = Lmode++ >1 ? 0: Lmode++;
          }

        if(digitalRead(RBUT == HIGH))
        {
          Rmode = Rmode++ >1 ? 0: Rmode++;
          }

        if(digitalRead(LFOOT == LOW))
        {
          d1 = getDistance(SLTP,SLEP);
          f1 = getData(Lmode,d1, dmax, 120,5000);
          f1 = f1 >5000 ? 0: f1;
          }

        if(digitalRead(RBUT == LOW))
        {
          d2 = getDistance(SRTP,SREP);
           v = volume;
           volume = getData(Rmode,d2, dmax, 20,80);
           volume = volume >80 ? 20: volume;
           changeVolume(v,volume,LCS, LINC, LUP);
          changeVolume(v,volume,RCS, RINC, RUP);
          }

        play_tone(ls,f1);
        play_tone(rs,f1);
        }
```

The Mark II did not operate, and I am unsure if there was an issue while soldering the components together or if one of the components did not work since I did not test them beforehand. As a result, I decided to remove the dynamic volume feature and get a version of the UST that was working. Thankfully, the Mark II was separated into subcircuits which made the Mark III easier to put together on short notice.

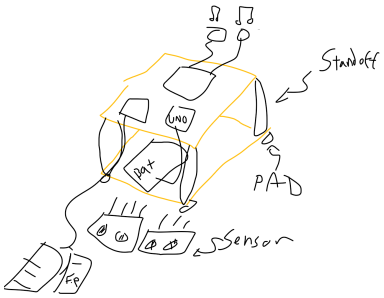


[3] X9C102, X9C103, X9C104, X9C503 Digitally Controlled Potentiometer. Renesas Electronics Corporation.

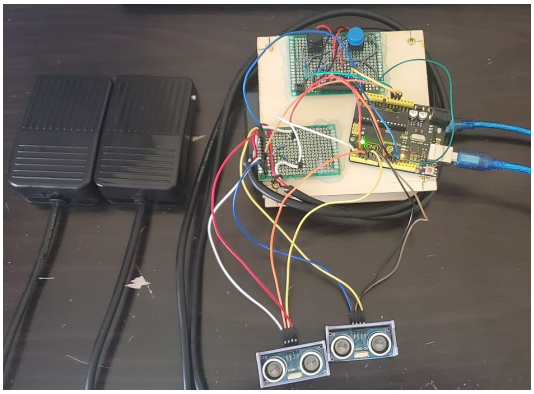
Part V: The Mark III



For the Mark III, I had redesigned some components to function better as a unit. The chassis would be two wood squares that would connect with four standoffs. On the lower level, the battery and two speakers reside, while the circuit boards are on top. Due to the weight of the foot pedal wires, I moved the foot pedals to their board and used a hot glue gun to keep it in place. I also moved the buttons to the speaker board and hot glued that board as well. I did not use hot glue on the Arduino Uno since I plan on reusing it in future projects. I reused the sensor chassis from the Mark I and chose to keep the sensors apart. To prevent movement, I placed furniture pads on the bottom.



There are still some minor design flaws with the Mark III's physical implementation since the foot pedal wires are still very long, so I wrapped them around the chassis. I should have cut them earlier in the design process to help alleviate this. The wires of the sensors tend to move the sensors, which can be annoying when you are trying to prevent one from being activated accidentally. The function buttons are also in an odd location since they are far from where the hands would be during most of the instrument's operation and have the speaker wires near them.



The Ardunio repeatedly calls the *TwoSpeaker()* function, similar to the *OneSpeaker()* function of the Mark II. The mode for the frequency calculations will change when the function buttons are pressed. When you step on the foot pedals, the frequency they represent will not change. I made a YouTube video to showcase how the UST works in real-time. Link: https://youtu.be/HnO8fP1xdik

```
void TwoSpeaker()
    {
    if(digitalRead(LBUT) == HIGH) //Cycles through modes on button
    {
      Lmode = Lmode++ >1 ? 0: Lmode++;
      }

    if(digitalRead(RBUT) == HIGH) //Cycles through modes on button
    {
      Rmode = Rmode++ >1 ? 0: Rmode++;
      }

    if(digitalRead(LFOOT) == LOW)//If High, I have pressed the pedal and do not want to update the tone
    {
      d1 = getDistance(SLTP,SLEP);
      f1 = d1 <2000 ? getData(Lmode,d1, dmax, 120,5000) : f1;
      }

    if(digitalRead(RFOOT) == LOW) //If High, I have pressed the pedal and do not want to update the tone
    {
      d2 = getDistance(SRTP,SREP);
      f2 = d2 < 2000 ? getData(Rmode,d2, dmax, 120,5000) : f2;
      }
    play_tone(ls,f1);
    play_tone(rs,f2);
    }
```

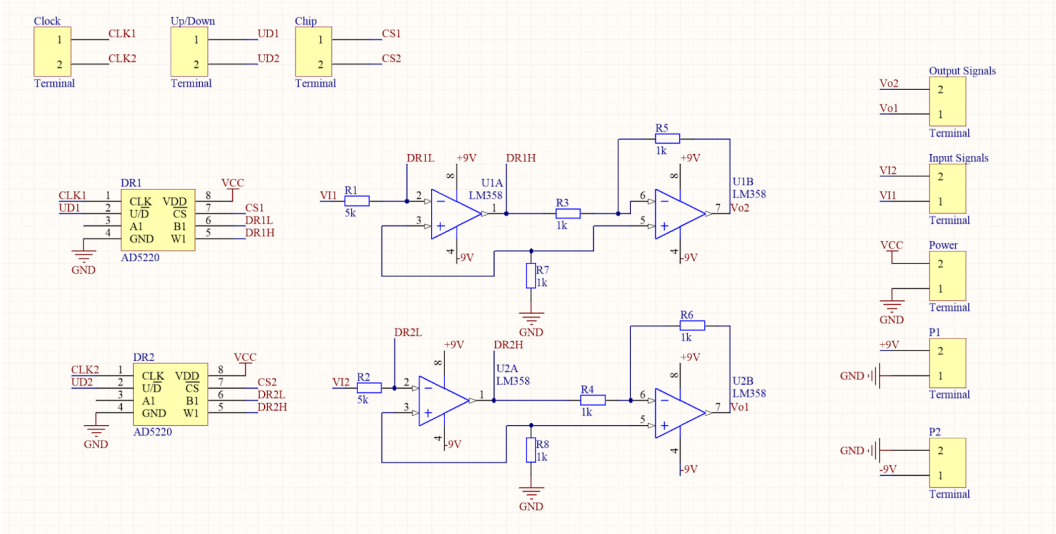
When playing the UST, I noticed that the buzzers do not sound pleasant

when playing at a constant tone, which disincentivizes the use of the foot pedals to hold a frequency for long. Additionally, I was having trouble discerning the jumps between exponential to linear, but I noticed the higher pitch in the logarithmic mode. I also was unable to perform rapid movements to change the frequency due to the loop processing time of the Arudino, which was limiting at times. Overall the code functioned as intended, and by this point, I had tried to remove any irregular operations.

The most problematic aspect of the UST was the learning curve for playing the device. Maybe if I came from a music background, I would be better acclimated to playing the instrument, but I found it difficult to move my hands in different motions simultaneously. I created a video trying to show the different ways I tried to play the instrument. Link: https://youtu.be/HnM9CCmMo1A

Part V: The Mark IV

I have begun work on the iteration of the device I will call the Mark IV, but it will not be completed before the project is due. The Mark IV will use different digital resistors, a different sound output device, and a printed circuit board. To design the circuit board, I am using the program CircuitMaker to place all the components on the schematic and then develop the PCB.



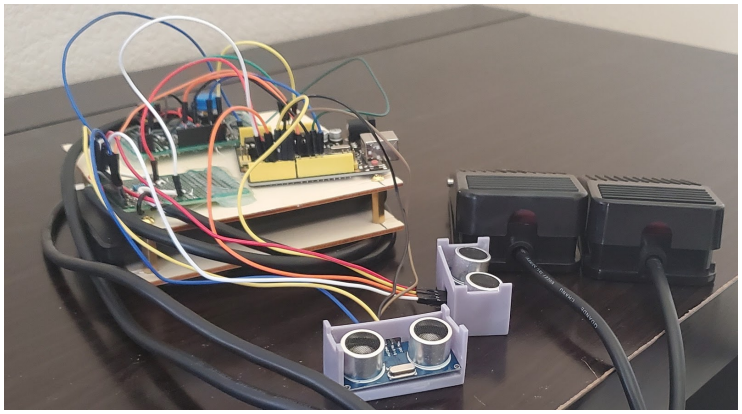
The Mark IV is only a work in progress and will be a project for me to tackle over the summer as a side hobby.

Part VI: Conclusion

The biggest takeaway from the UST was that reconceptualization was harder than I imagined, but I loved slowly improving my design. One of the more difficult things about the UST was learning how I would play it. Initially, I sought out to recreate tunes I was familiar with, but due to both hardware limitations and how I designed the inputs, I was unable to create the sounds I wanted. I had to change my expectations for the sounds I would hear and want to output, which made the thought process slightly easier.

In the musical side of the experience, I got a better feel for the process of creating anything relating to music. Coming from a non-musical background, aside from a couple of months in 4th grade, it was interesting incorporating my major and the class. Having little direction for the project made it the most interesting to develop solutions to the problems I created and add features to the UST. I also used some of the hardware and software design skills I have learned in my major curriculum, even though the hardware may not have been fully successful. Soldering was also a little rough, but I had gotten better with more practice the more time I spent trying to solder the components to the perf boards.

Overall, I was successful in my proof of concept, but I was unable to achieve every goal I had set out for myself. However, I tried to set a large number of goals so that I always had a new thing to strive for, and I am still shooting for those goals now. If I had started the project with what I know now, I would have taken a slightly different set of steps, such as starting with a design for a printed circuit board, but I am happy with the project and look forward to future projects that utilize the skills I have improved through the development of the UST.



Work Cited

Arbi Abdul Jabbaar (September 19th, 2019) Ultrasonic Sensor HC-SR04 with Arduino Code for Ranging Test (Version 1) [Source Code]. <https://create.arduino.cc/projecthub/abdularbi17/ultrasonic-sensor-hc-sr04-with-arduino-tutorial-327ff6>

Brett Hagman & Fotis Papadopoulos (January 13th, 2017) Tone Library (Version 1, Revision 14) [Source Code Library]. <https://github.com/bhagman/Tone>

X9C102, X9C103, X9C104, X9C503 Digitally Controlled Potentiometer. Rev. 4. 2019. Renesas Electronics Corporation. January 11th, 2019 <https://www.renesas.com/us/en/document/dst/x9c102-x9c103-x9c104-x9c503-datasheet>