**Microphone Array**

Design Documentation

CSE 453 SPRING 2016

Thomas Schouten

Korben Grover

Arunan Bala Krishnan



**University at Buffalo**

***The State University of New York***

# **Contents:**

## **1. Introduction**

* Purpose of the Array
* Scope of this project
* What we have working

## **2. Design Overview**

* Approach to the task
* Constraints and Solutions
* Algorithms
* Filtering
* Localization
* System setup
* Amplifier Connections
* Storage and Data model

## **3. Continuing the Project**

## **4. Parts List**

# **Introduction**

* ***Purpose of the Array***

The purpose of this project is to record and track birds as they interact with each other. There are many unknown social interactions between birds, and it is hard to keep track of their locations as they call to each other. Thus, a method of triangulation of sound sources is needed to keep track of these bird calls. To perform this, a microphone array is to be used, with data from each of the microphones working in order to facilitate this task.

* ***Scope of this Project***

The scope of this project is largely limited to time. This was to be worked on over one semester, including design, construction, implementation, and testing. As such, limited time available for several of the tasks means that the project is likely to be in an unfinished state, but what should be presented is a functioning prototype that demonstrates the idea of the project, and especially to show what full functionality may be capable of.

* ***What we have working***

Here at the end of the semester, further work on the project must be halted, and presentation of what currently stands is the next step. Many trials and issues held up work on this project, leading to an end product below including the plans to continue the project. What has actually been done represents the original plan, but cannot be accomplished do to the hardware available.

The microphone array construction features 3 microphones, capable of providing two dimensional triangulation. A low sample rate affects what can be detected by these microphones, which largely limits the accuracy of this triangulation. In fact, the system is only able to tell which side of the x axis the sound originated from. On the software side, basic logging and analyzing of the data is being done which could easily work with higher sample rates. The original plan was to do calculations after recording data because the computations would be too great for streaming data. A viewable graphic is also provided, showing a representation of where the system heard the sounds coming from.

This system was designed with much more in mind, and can thus be easily expanded upon. Many pieces of hardware exist to support a much higher performance, but missing key components has limited the state of the project to what it currently is.

# **Design Overview**

* ***Approach to the task***

Faced with this project, we decided to use a grid of microphones, linked into an ADC (Analog to Digital Converter), which attaches to an arduino to finally deal with the data in software. Many challenges lay in completing this implementation, as can be seen in the next section:

* ***Constraints and Solutions***

The first hitch in this project, was the ADC. The plan was to use a simultaneous sampling ADC with multiple channels. This way all of the data from the microphones would be sampled at the same time. We ended up with the LTC2345-18 for this purpose. In order to interface with it, we needed an appropriate microcontroller as well. However, the first microcontroller, the Linduino turned out to be too slow, with a sample rate of only 100 Hz at most. In order to get higher sample rates we used an arduino and it’s ADC along with three microphones to get data. The sampling rate of around 2kHz is dramatically lower than we originally planned. The microphones also are no longer sampled at the same time. However this is a start at proving the concepts work.

Another concern was what to do with all the data being gathered. The current solution is to dump it into a log file on the computer as soon as it is available. This allows for less overhead on the Arduino, as it only has to pass the data on to the computer. Two things keep this system slow, are the sample rate of the Arduino ADC ports and the USB connection. In order to expand to a full eight microphone setup, the data will likely need to be stored on an SD card, which is a slightly faster alternative to the USB connection. Further work on integrating the ADC with the microcontroller can result in a better, more accurate sample rate as well.

* ***Algorithms***

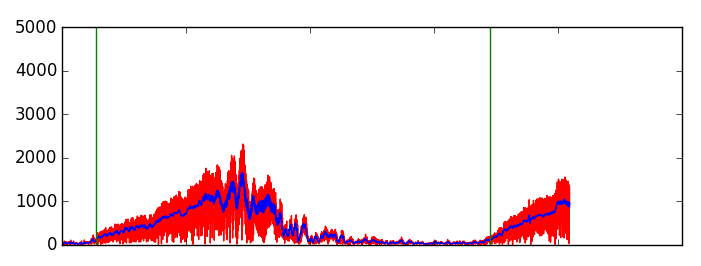
Several Algorithms are used in this project, primarily to work with the data afterwards. Since the hardware is focused on simply dumping the data onto the computer, this section will focus on what algorithms are used in post-processing of the data. There are two primary sections of this: Filtering of the data, specifically to determine when a sound has occurred for each microphone stream, and then localization, using that time data to determine where that sound originated from.

* ***Filtering:***

This algorithm starts by taking a low-pass filter by taking the average over the absolute value of the data. This gives a smoothed magnitude version of the sound data. After that, a cutoff is specified, where sounds above this cutoff represent a sound. The algorithm steps through the data, keeping track of where the data moves from a state of no sound, to a state of active sound.

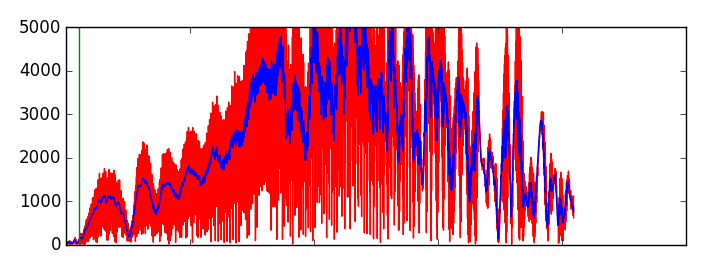
To reduce the effect of background noise and random jumps and dips, additional buffers are defined to determine the length of above cutoff needed to start recording, as well as a minimum drop-off needed to prevent small dips from splitting a sound into multiples.

From here, it is mostly tweaking the size of these cutoffs and buffers until a good representation of when sounds start is obtained. Image 1 and 2 below are a few samples of this algorithm in work, for a pre-existing sound file of a bird chirping.



*Image 1*

The Red area in image 1 represents the original sound data, with the blue data being the low-pass filtered version of said data. The vertical lines show where this algorithm has detected a sound beginning.



*Image 2*

Image 2 shows that the algorithm works despite a highly noisy signal, by being able to look over sudden dips or growths that dance around the cutoff value.

* ***Localization:***

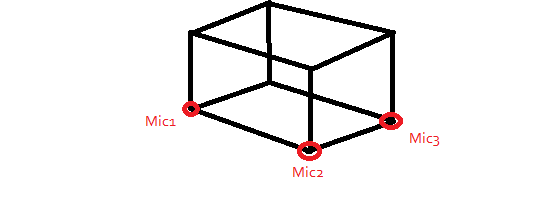
The logic behind the algorithm used to find the locations of sound sources largely comes from the article “A Python Implementation of Chan’s TDoA algorithm for Ultrasonic Positioning and Tracking”. What we had to do with this was adapt it into our code such that it can interpret the data we have. The biggest change required was to move from a 3D system into a 2D system, as we were only able to get 3 microphones recording.

One thing worth noting is that this algorithm can return either 0,1,or 2 positions for every time sampling. For 0, we obviously just throw it out. For two, we decided to keep just the one further from the array, as it is the more likely option.

Should someone work on adding more microphones and dimensions to this project, the matrix micPos should be updated to contain the relevant information on the position of each matrix, and an appropriate 3D algorithm should be implemented instead.

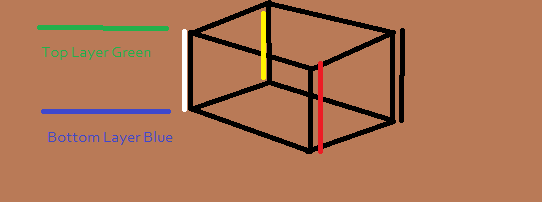
* ***System Setup:***

The microphone array is a structure of PVC piping, about 1x2x1 feet in dimension, standing atop one foot stands. It was originally planned to feature 8 microphones on it, one at each junction of pipes. As of now, there are only three microphones hooked up to the structure shown in Figure 1.



*Figure 1 - Skeleton structure with three microphones*

In order for debugging the wires are color coded with a single color or two colors. The single color band wires deliver the same signal to each microphone such as ground and power. Each microphone has a personal wire, featuring two colors to determine what microphone it connects to. Figure 2 shows the mapping of the two color wires to their respective corners. Table 1 and Table 2 show the full color scheme and the wires respective properties. Notice that the figures and tables are set up for eight microphones but currently only have three.



*Figure 2 - Mapping of two color wires*

**Table 1. Analog input wires**

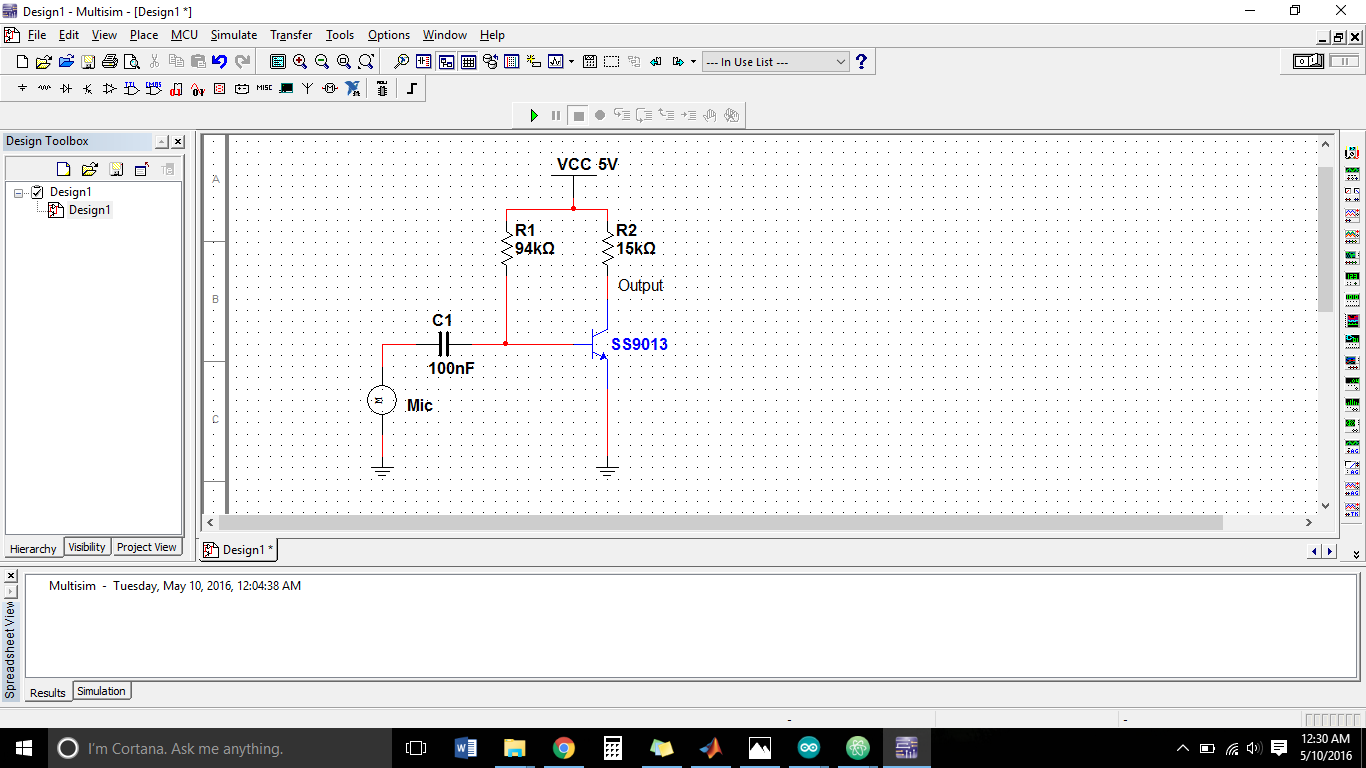
|  |  |  |  |
| --- | --- | --- | --- |
| **Microphones** | **Individual Wiring** | **Bottom Wiring** | **Connected** |
| 1 | White | Blue | A0 |
| 2 | Red | Blue | A1 |
| 3 | Black | Blue | A2 |
| 4 | Yellow | Blue | Not Connected |
| 5 | White | Green | Not Connected |
| 6 | Red | Green | Not Connected |
| 7 | Black | Green | Not Connected |
| 8 | Yellow | Green | Not Connected |

**Table 2. General wires**

|  |  |  |  |
| --- | --- | --- | --- |
| **Group Wires** | **Terminal** | **Used in Implementation** | **Pin on Arduino** |
| Red | +5V | yes | 5V |
| Green | -5V | no | - |
| Blue | Ground & GNEG & COMM | yes | GND |
| White | GPOS | no | - |
| Yellow | Not used | no | - |

* ***Amplifier Connections:***

In order to amplify the microphone signal we used the SS9013 BJT in a common emitter circuit as shown in Figure 3. The gain of the circuit was measured to be 60 V/V with a continuous 1 kHz signal. The original plan was to use AD603 amplifiers to amplify the circuit which uses a reference voltage from 2V to -2V to control the gain of the signal. Using the AD603 would allow for software to control the gain of all the amplifiers.



*Figure 3 - Amplifier circuit diagram*

* ***Storage and Data model:***

All recorded data is stored in a specific txt file to ease the process of analyzing the said data. The txt file is a result of running “readClient.py”, a python algorithm that helps capture data. This txt file, named “dataLog.txt” is stored in the root folder and it contains microphone data for each of the three microphones. It is advisable to rename the file after a run as further tests will overwrite the existing file. The data for each microphone is stored in a way where they are separated by commas and data from a new cycle will be stored on a fresh line. Once all data has been recorded, the last line of the file will state the sample rate of the system during the run.

# **3. Continuing the Project**

Seeing as how several parts arrived the day before the proper presentation, we were unable to set up substantial parts of the system that would allow this project to be much more effective at its task. A general checklist of things to work with still is as follows:

* The faster 890 board arrived, allowing for a much faster read off of the ADC. Thus, the microcontroller currently being used should be removed, and replaced with the ADC attached to the 890. However, due to the difference in operation, new code will have to be written to work with the ADC. Also, the ADC itself will require an input clock signal to drive it.
* Additional microphones can be added to the array. The wiring permits up to eight microphones on the array. One must setup the proper circuitry for each one, and make sure to extend the common wires across to access microphones further down the array. With the microphones in place, their respective signal wires can be routed into the ADC.
* The current power that comes straight from usb cord connected to the computer creates a lot of noise on the ADC. There are voltage regulators that we got but had no time to install them, these might reduce the noise seen on the ADC. Also the ADC board has differential amplifiers at the inputs which will reduce some of the noise, the Linduino does not have these differential amplifiers.
* There are also voltage controlled gain amplifiers that we did not install because the regulators came in the day before the project was due. They require a +5V and a -5V power supply hence the reason for needing the voltage regulators.
* At this point, one should first make sure that audio can correctly be recorded from all the microphones. Once proper log data can be acquired, the algorithms can be updated to work with this new data.
* The filtering algorithm should be mostly the same. Near the top of the getStartTimes method, a few commented out lines hide a graph that show the data being worked on. You can uncomment these to see the signal, and then adjust the parameters of the filtering as needed to get the results desired.
* The localization algorithm currently works in 2D space, but can easily extend to 3D space as it is. The biggest change that would happen is the micPos array, in which the positions of the various microphones are hard-coded in x-y format. Turning to 3D would put it in x-y-z format, and then all the additional entries that more microphones would entail. The rest of the code should work, once edits are made to account for this additional dimension.

# 

# **4. Parts List**

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Part #** | **Amount** | **Received** |
| Demo Board | DC2326A-A | 1 | yes |
| Demo Companion Board | DC2026C | 1 | yes |
| Data Acquisition Board | DC890B | 1 | no |
| Microphone | CMC-2742WBL-25L | 8 | yes |
| 14-Pin Ribbon Cable | CA2440 | 1 | yes |
| 20 AWG Wire |  | ~100’ | yes |
| Device Frame Pipe | PVC ½” | ~10’ | yes |
| Top Corners | 3-Way ½” connectors | 4 | yes |
| Bottom Corners | 4-Way ½” connectors | 4 | yes |
| BJT | SS9013 | 3 | yes |
| Microphone (Used) | Arduino Microphone | 3 | yes |
| Through hole PCB |  | 1 | yes |
| Assortment of resistors & capacitors |  |  |  |