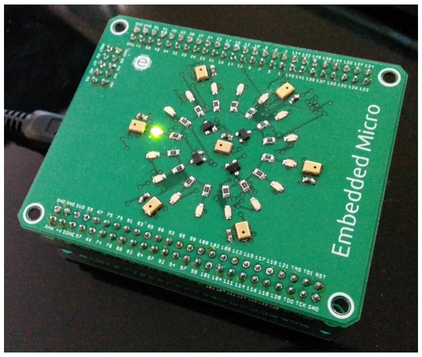
Detecting the Direction of Sound With a Compact Microphone Array

Purpose

The goal of this project was to detect the direction of potentially multiple sound sources using a compact array of microphones. The idea is that this could be used to give robots a sense of hearing. I built a board with seven microphones on it that is able to sample each microphone simultaneously. With the audio samples I was able to detect multiple audio sources given that their frequencies are sufficiently different.

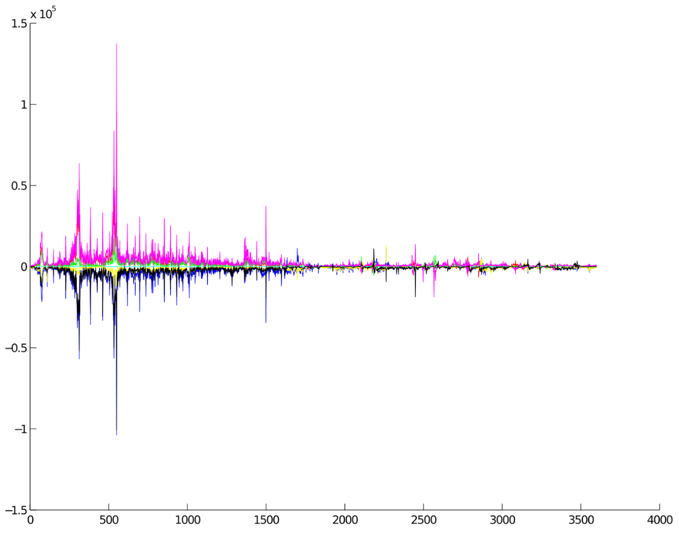


Data

The data consists of short audio samples from all seven microphones. I captured these samples using an FPGA to interface with the microphones and a small Java program to interface with the FPGA. The samples were saved in CSV files that could be imported into Matlab. The samples were split up into many smaller clips to get more training data. The microphone ring has roughly a 1 inch radius.

Pre-processing

My approach to solving this problem was based on the idea that I should be able to calculate the delay relative to the center microphone for each frequency in each of the audio streams. This proved to be a little tricky to get right. I first normalized and set the mean to zero for the audio streams. I then used a Hann window and calculated the FFT. From the FFT, I extract the magnitude and angle for each frequency. I then calculated the difference in each angle with the angle of the center microphone. This angle is assumed to be between +/- pi. This is ensured since the microphones are close to each other, for audible frequencies the difference is guaranteed to be in this range. The difference in angle is then divided by the frequency to calculate the delay. I also dropped the lowest 0.5% of frequencies and the highest 40%. The resulting range is roughly the audible signal. The following figure is what the pre-processed data looks like.



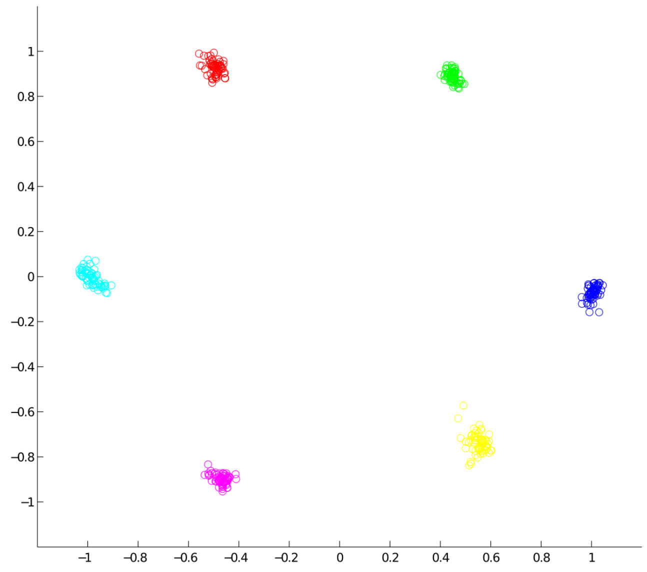
You can see that the pink signal has the most positive delay and the blue has the most negative delay. It is also obvious that these delays are shared across all frequencies with

rajewski@stanford.edu

substantial magnitude. This is because this is a sample from a single source.

Delay to Direction

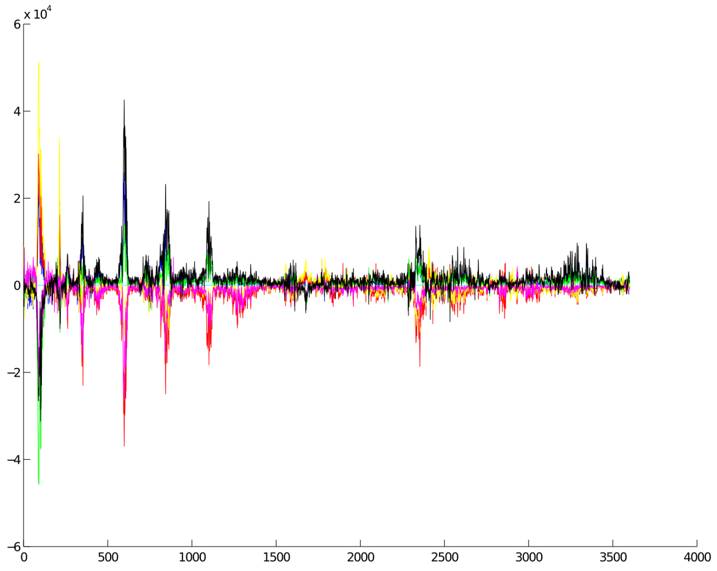
The first part of my project I needed a model to convert the relative delays to a predicted direction. The delays I used to train were an average of the delays of each frequency in the sample weighted by the magnitude of that frequency. The averaged delay was then normalized to have it's largest component have a magnitude of 1. I started by trying to come up with a way to do a regression on the angle directly. This proved to be really difficult due to the angle's modulo nature. I tried a few variations on linear regression that failed before coming up with the idea to predict a point on the unit circle instead of the angle directly. I then trained two linear models, one to match cosine of the label (known angle), and one to match sine of the label.



This approach worked very well and the figure to the right shows the output on the test data. Each color is a different label. The 6 test locations were evenly spaced at 60 degree increments. If you imagine the microphone array as being in the center of the plot, the points are roughly where the sample was taken from (the distance is not detected).
By using this method, I was able to accurately approximate the direction of a single source (or the average direction of multiple sources).

This model also gave me a way to tell if the direction was ambiguous. If the predicted point wasn't near the unit circle, either close to the center or too far out, I knew that the delay signature didn't correspond to a single source. This was important for the second part of my project.

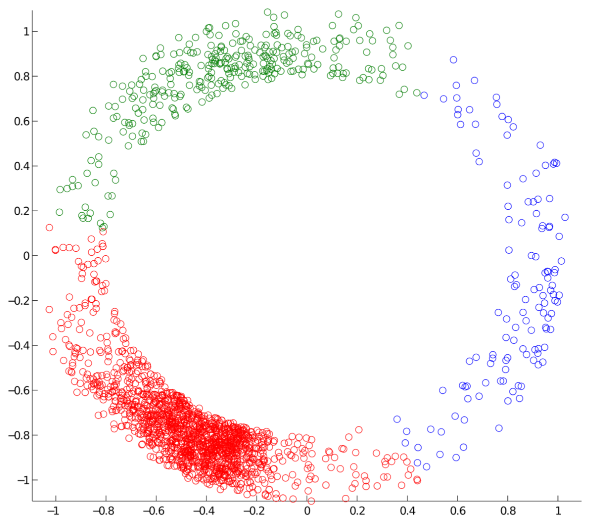
Detecting Multiple Sources



When I started this project I wanted to be able to detect multiple sources if I was able to successfully detect at least one. The approach I decided to take was inspired by looking at some of the preprocessed data similar to the figure on the right. This sample consists of three audio sources (music, talking, and tapping on a coffee cup lid). If you look closely at the very low frequencies you will see a region where the pink signal is on top. There is then another region with the yellow signal on top and

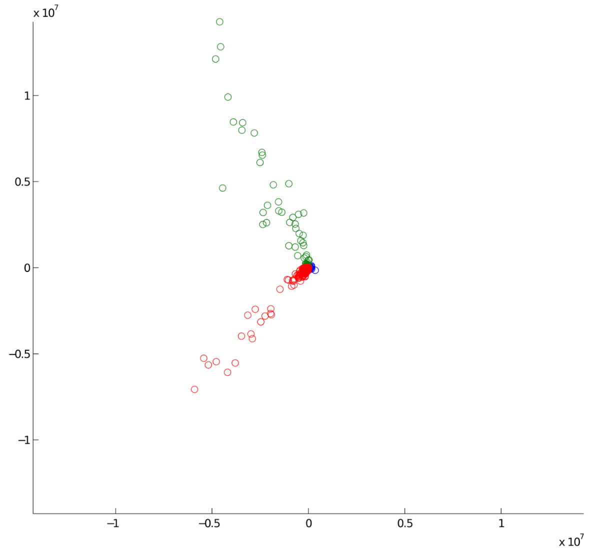
Justin Rajewski
rajewski@stanford.edu

finally a large region with black on top. These correspond to the different sources. I realized I could then take each frequency and, using my model from before, predict the direction for each frequency individually. The figure on the right shows that output with the points that are not close to the unit circle removed. The removed points don't correspond to a predictable direction and just add noise. The colors roughly correspond to the each source and are from the final output of my algorithm.

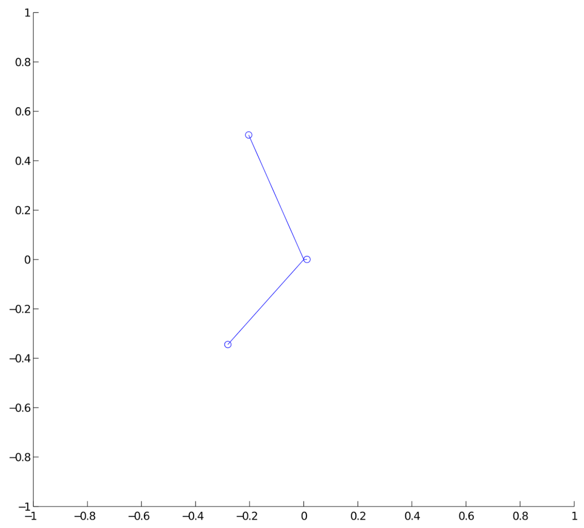


You can see there are a lot of red points. These are from the frequencies where black is on top. The green points are from frequencies with yellow on top and the blue is the ones with pink on top.

My plan was to run k-means on the data to find these clusters, but I also wanted frequencies with larger magnitudes to matter more. I first weighted each point with its corresponding magnitude, but I found that using the magnitude squared gave me better performance when trying to discern multiple sources. The graph on the right shows the points multiplied by the magnitude squared. It is now obvious there are at least two sources. The third source (tapping on the cup) is much quieter, but still detectable.



By running weighted k-means I was able to get find the directions of each source. I then took each direction and weighted it by the relative magnitude (calculated by the percent of the total weighted points belonging to it). The result is the graph on the right. You can see the three sources with the right pointing source designated much quieter than the other two, but still pointing the correct direction.



I now had an algorithm that worked well for picking n sources from a signal. However, I didn't want to have to specify n.

Justin Rajewski
rajewski@stanford.edu

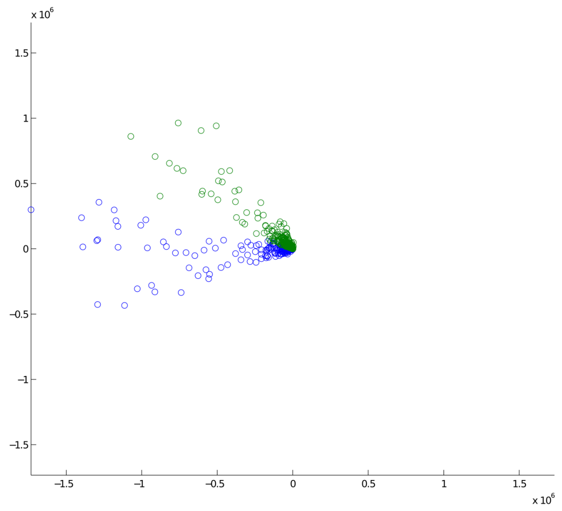
Detecting the Number of Sources

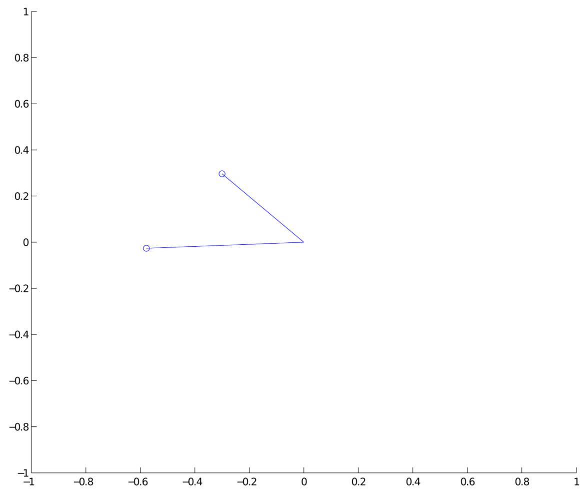
The problem I had with k-means is that you need to specify the number of sources. If you specify too many you will likely split a source into two or more where the average of those is likely the real source. If you specify too few, you end up with predicting the source is between multiple real sources. To combat this I created another algorithm around the weighted k-means I was using.

The algorithm starts by running k-means with only one centroid. Each iteration it adds another centroid and tries again. After each attempt it looks at the sources that were found. If these sources are likely to be from the same real source, the algorithm stops and uses the previous iteration's results.

The only trouble with this was coming up with a way to know if two source are likely to be from the same real source. After watching k-means work on different samples, I realized that if the difference in the angle was small and the difference in magnitude was large, it was likely that they came from the same source. This is due to the nature of each source typically being a relatively narrow spike.

My first attempt at this I used the absolute value of the difference in the angles divided by the absolute value of the difference in magnitude. If the lowest value between any two centroids was less than a threshold, it stopped. I fiddled with this for some time with decent results. However, I realized that I was weighing the fact of two signals with similar magnitudes were likely to be different signals too heavily. That lead me to my final criteria $d = \frac{|angleDifference|}{|mag_1 - mag_2| + 30}$. I added the 30 term to the denominator to put less emphasis on the difference in magnitude. I also added a threshold for the minimum acceptable magnitude. After playing with the threshold again, I was able to get this to work on all my test cases including the following with two relatively close sources. The two sources were at 180 and 120 degrees.





Justin Rajewski
rajewski@stanford.edu

Results

Model	Samples	Error (degrees)	Error (%)
Linear Reg Training	1920	2.6	1.44%
Linear Reg Test	384	3.4	1.89%
K-Means Test (single, w=mag)	30	4.8	2.67%
K-Means Test (multi, w=mag)	5	7.7	4.28%
K-Means Test (single, w=mag^2)	30	5.3	2.94%
K-Means Test (multi, w=mag^2)	5	6.1	3.39%
K-Means Test (single, w=mag^3)	30	6.9	3.83%
K-Means Test (multi, w=mag^3)	5	6.1	3.39%

The linear model worked very well and the error is easily within the amount I could have moved while using my phone to play music from the various angles.

For the k-means model, I tested it with various different weights and found using the magnitude squared resulted in the best overall performance. I found it interesting that the error for a single source was better with lower powers of the magnitude, but the error for multiple sources was better with higher powers. I believe this is because it simply brings out the loudest frequencies more which usually overlap less between sources giving a clearer indication of the direction.

For the multi point test cases, I used two sources for each. These samples were more difficult to work with and I didn't split each sample into many small clips like I did with the linear model since using long clips gave higher frequency resolution and better performance. This is the reason for so many less samples.

Future

One of my goals for this project was to eventually move my algorithm to the FPGA on my audio capture board so that the direction could be detected in real time. However, I didn't have time to explore this and the current k-means algorithm would be difficult to port to an FPGA. I would like to explore simpler ways to get the same kind of performance.

I would also like to repeat everything with only three microphones. Using three is theoretically possible and I would like to know what kind of performance improvement (if any) you get with more than three microphones.

Justin Rajewski
rajewski@stanford.edu