5-1-2006

Sound Source Localization and Separation

Biniyam Tesfaye Taddese
Macalester College, btaddese@macalester.edu

Sound Source Separation and Localization.

Honors Thesis in Computer Science

Biniyam Tesfaye Taddese.

May 1st 2006.

Macalester College
St. Paul, Minnesota

First Reader:
Professor Daniel T. Kaplan,
Department of Mathematics & Computer Science

Second Reader:
Professor G. Michael Schneider,
Department of Mathematics & Computer Science.

Abstract

People face the problem of sound source localization and separation in situations where they attempt to localize and focus on a source of sound among a dissonance of conversations and background noise. This paper synthesizes a sound source localization routine. We utilize a general source separation technique, Independent Component Analysis.. Particularly, basic ICA was applied to separate mixtures of low frequency, narrow band, non-Gaussian signals by using closely spaced uni-directional microphones. The localization routine worked with an average condition number of 10. The routine was tested on data collected in a laboratory.

Acknowledgements

I would like to extend my sincere gratitude towards Professor Daniel Kaplan and Professor James Heyman who provided guidance and supervision throughout this project. I also would like to thank the Macalester Keck Foundation, which has funded my research in Summer 2005.

I thank my parents, Tesfaye and Fetlework, whose life story instilled a positive energy and a sense of responsibility in my soul. I am very grateful to the efforts of my siblings who home schooled me as I struggled with child-sickness, which hindered my regular school attendance: my eldest brother, Zelshe, taught me how to write; my brother Muluye and my sister Menbi (Dmet) taught me how to count. This Honors Thesis in Computer Science would never be realized without the help of all these people.

Table of Contents:

Chapter 1

Introduction

1.1 The Cocktail Party Problem

People are generally able to discern the direction that a sound is coming from using two ears. The combination of the slightly different signals that arrive at the ears enables us to deduce, intuitively, the direction of the sound. Similarly, a biologically inspired sound localization system can be built by making use of an array of microphones, which are hooked up to a computer. In addition, such a system of microphones can be made to extract any particular sound from a mixture of sounds produced simultaneously by several sources. Such a situation is reminiscent of the "cocktail party problem", in which a person attempts to focus on a single speaker among a dissonance of conversations and background noise.

Source Localization is a very well established technique that has a wide range of applications. Blind signal separation is still a very current area of research, with no generally acceptable solution. My project consisted of studying the benefits and limitations of these existing techniques, and applying them to solve the problem of sound source separation and localization.

1.2 Source Localization Techniques:- The GPS

The applications of source localization techniques range from remote sensing to the Global Positioning System. The mathematical details of these applications are basically variations of a triangulation scheme: 1. using multiple sensors, which detect a signal emitted bythe source to be localized, or 2. using multiple emitters, whose signals is sensed by the sensor (eg. GPS receiver) to be localized.

In this project we attempt to localize a single sound source by using four microphones, which are placed at the tips of a possibly irregular tetrahedral. The minimum number of microphone needed depends on the dimension of the space in which the source is constrained to, such as a line or a plane. In an ordinary situation, the source can be located anywhere in the three dimensional space. Hence, as will be elaborated later, a minimum of four microphones is needed to accomplish the localization of a

source to a point. The use ofmore microphones would increase the accuracy of the localization of the source at the cost of computational time; however, we shall see that the marginal benefit of using more microphones than the minimum required is not worth the cost.

1.3 Blind Signal Separation

The need for Blind Signal Separation arises in many situations. For instance, electrocardiogram sensors record a mixture of pulses from different parts of the body. One needs to be able to extract the individual source pulses, which resulted in the detection of mixtures of signals at the surface of the body. Such problems demand the development of blind signal separation techniques.

Unlike some other signals, there is no general technique that is applicable to blind sound source separation. [1] The separation techniques which claim generality are overly complicated and do not necessarily perform perfect signal separation. Therefore, we intend to limit this project to a semi-blind sound source separation problem.

We have studied the feasibility of various source separation techniques. Independent Component Analysis is the major technique that is considered. The technique assumes that the source signals, which constitute the observed mixtures, are independent and identically distributed (i.i.d.). Hence, it assumes a particular probability density function for the source signals. Despite its applicability in other fields, this method cannot be applied directly to audio separation for reasons that will be discussed in *Chapter 3.4*. In the past decade, researchers have come up with various advanced extensions of Independent Component Analysis. Some of these extensions are studied. However, we have tested a more original way of implementing basic Independent Component Analysis by making use of closely spaced uni-directional microphone clusters with certain restrictions, which are discussed in *Chapter 3.6*.

1.4 Overview of the Project

The geometry of sound localization problem was studied before building a robust triangulation routine, which relies on a normalized cross correlation routine that computes delays between arrival times of signals at different microphones. The sound

localization routine was tested thoroughly, and worked as desired. The testing involved a set up of the reverse problem. In the reverse problem we aspire to find the locations of each of the microphones; we do this by performing separate recordings of sources, which are placed at several known locations. This helped to estimate the sensitivity of the routine to misspecifications of location coordinates, which mainly stem from distance measurement errors.

A microphone cluster was set up in a lab with coordinates, where the microphones were placed at the corners of a tetrahedral. A sound localization experiment was done whose results proved the measured success of the sound localization routine written in MATLAB.

A significant amount of time was spent studying blind source separation techniques, particularly Independent Component Analysis. Among the extensions of ICA, the Lambert FIR matrix approach and Torkkola's neural network based approach were considered. Finally, we resorted to using the basic ICA developed by Bell & Sejnowski in 1995 with some modifications of our experimental set up, which made the sound separation process only a semi-blind one. The results of this experiment were successful despite the specificity of the case that we dealt with. As mentioned above ICA relies on an assumption regarding the probability density function of the source signals. Thus, the sensitivity of ICA to misspecifications of the pdf were also studied.

Chapter 2

Sound Source Localization

2.1 The Geometry of the Problem

Analyzing the geometry of the problem is important; because it allows us to address the following issues. Any source localization system can be prone to bewilderment regarding the location of the source due to aliases. Aliases arise when we do not have enough sensors or when the geometric placement of the sensors makes some of them redundant. The problem of aliases can be solved by adding more sensors to our localization system. However, consideration of the geometry of the problem allows us to add microphones economically. That is, we can determine the minimum number of microphones needed for any given situation.

In some cases, we may need to constrain the degree of freedom of the source of sound. This allows us to do simple experiments by using even smaller number of microphones than the number of microphones that we would need to localize a source in 3D. It is evident that no blind source localization can be achieved using one microphone. So, we start by looking at how we need to constrain our source of sound, assuming that we only have two microphones.

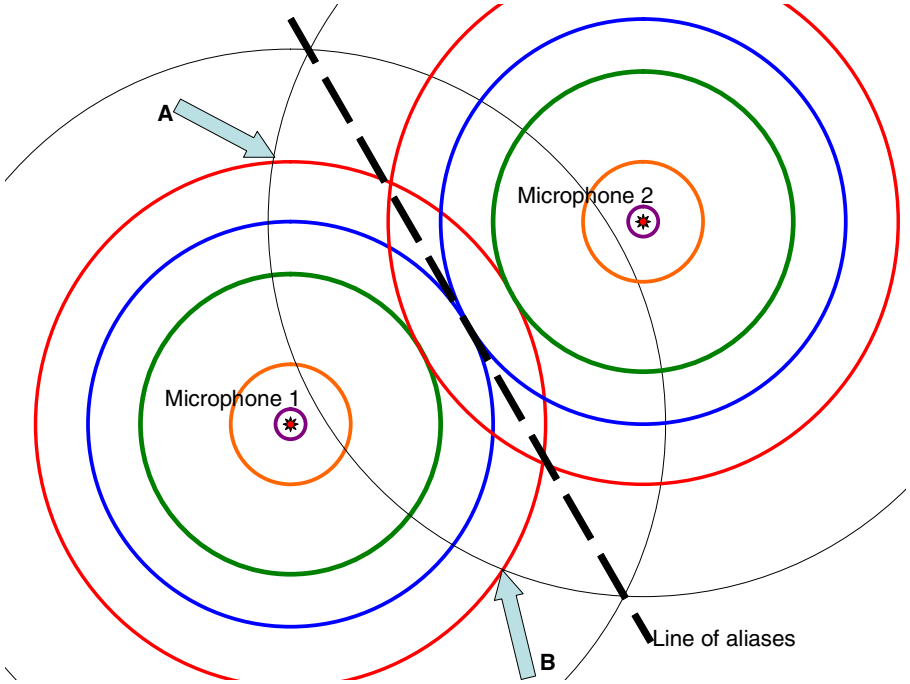

Figure 1Geometry of two microphones in a plane:- Points A and B are examples of aliases.

The only way that a source can be localized to a point using two microphones is if the source is constrained to a line that passes through the two microphones. If this constraint is lifted, the precision of the system degrades. *Figure 1* gives us some idea of the locus of points of aliases of two microphones and a source that is constrained to a plane that also contains the microphones; the black dashed line and points A and B on *Figure 1* are good examples of aliases. We generalize the depiction of aliases by using *Figure 2*, which is generated by a MATLAB program included in *Chapter 4.3.1*. The loci of points of aliases depicted in *Figure 2* are hyperbolas. Consequently, the loci of points of aliases which are associated with the localization of a source in 3D using two microphones are hyperboloids.

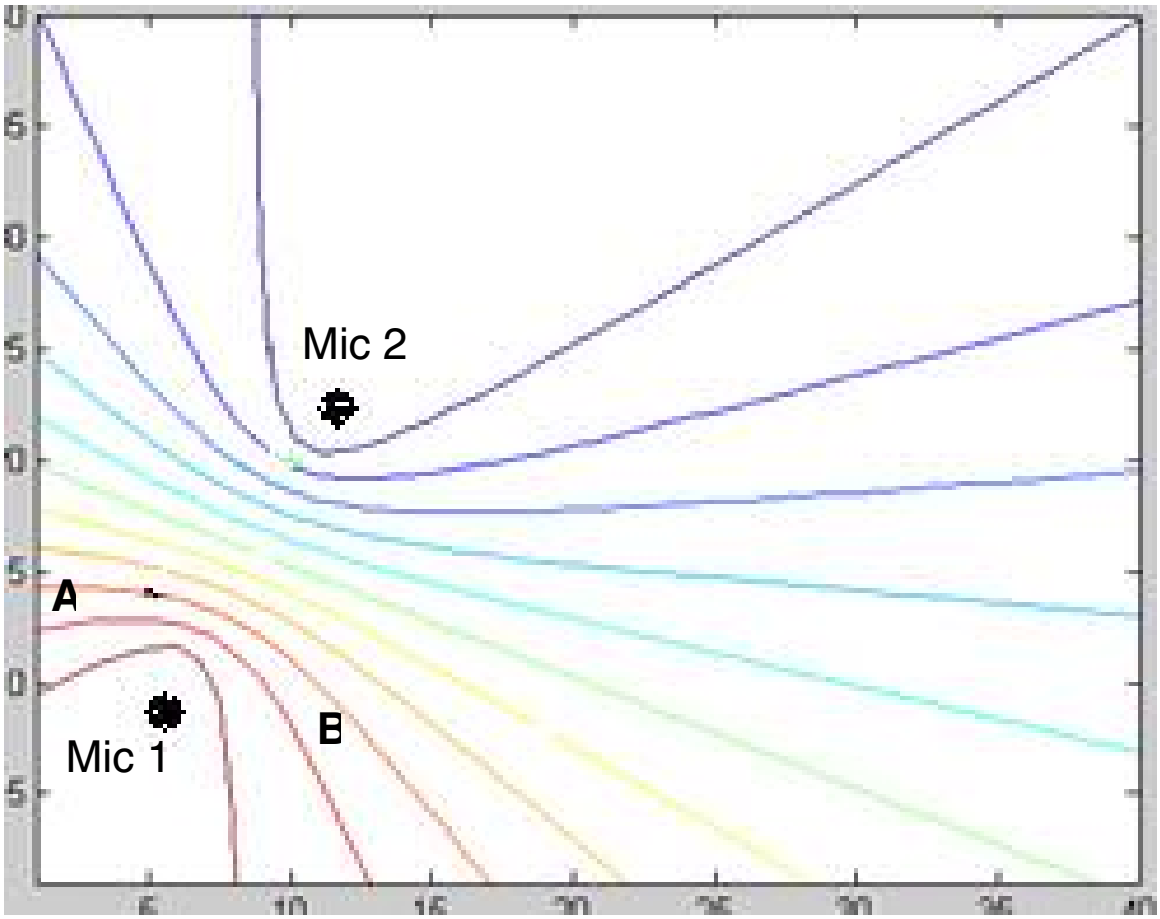

Figure 2 Locus of points of aliases of two microphones in a plane.

We consider adding a third microphone to our system; because, we want to ameliorate the constraints placed on the source. Such constraints were imposed to make the system precise to a point. Adding a third microphone on a line that passes through the

previous two microphones results in redundancy. Thus, the three microphones should not all be on a single line.

Three microphones placed at the corners of a triangle may seem to be adequate to localize a source to a point in a plane. However, this is not true. The location of the point would be determined as an intersection of three hyperbolas and/or planes. Such a system is over determined and a solution may not exist. The source localization system has its inherent measurement imprecision, which emanate from the limited sampling capabilities of our data acquisition system and the imprecision in measuring the locations of our sensors. The consequence of this imprecision and over determinacy is possible inconsistency of the system, which is illustrated in *Figure 3*.

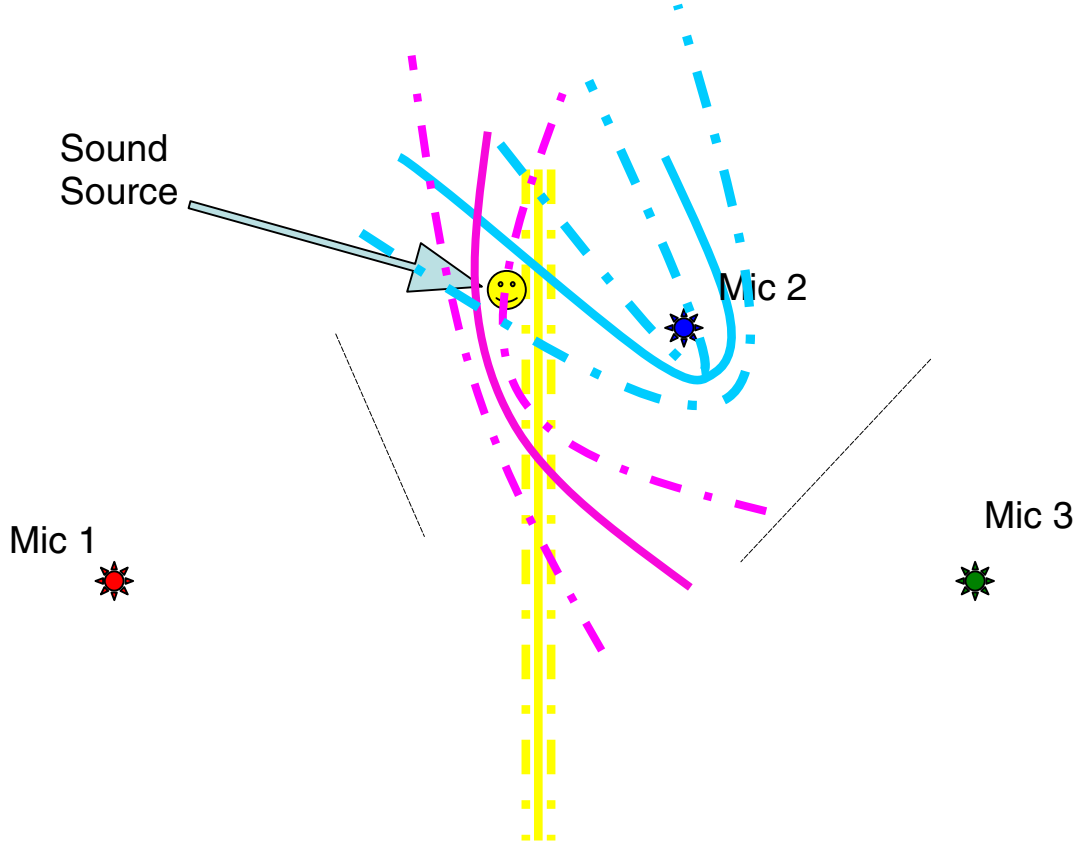

Figure 3 Inconsistency resulting from the imprecision and over determinacy of the system. Note the widening error bars, which result from the imprecision of time delay measurement for a source that is far away from pairs of microphones compared to their separating distance.

In *Figure 3*, each pairs of the three microphones make different claims about the location of the source. For instance, microphone 1 and 3 believe that the source is located on a line that perpendicularly bisects the line segment which connects them.Each of the

other two pairs of microphones believes that the source is located on the two different hyperbolas illustrated above. Optimization is needed to find a common ground for these inconsistent claims. The details of the optimization routine are discussed in *Chapter 2.3*. If we have 3 microphones and a source in 3D, then the system would only be able to localize the source to some simple closed curve, which resembles an ellipse, after optimization.

Having four microphones and a source in 3D would give us a situation similar to the one illustrated in *Figure 3*, except for its increased dimensional complexity. The line would be a plane and the hyperbolas would be replaced by hyperboloids. It is essential that the four microphones are placed at the tips of some sort of a tetrahedral in order to avoid redundancy of microphones. Placing all four microphones on a plane degrades the precision of the system by producing aliases, which we are trying to avoid.

2.2 Detecting Delays: Cross Correlations

The time delay of a signal between two microphones is the difference between the times of arrival of the signal at the two microphones. We studied the geometry of the problem by assuming that we know the time delay of the source between any pair of microphones. Cross correlation is a routine signal processing technique that can be applied to find such time delays by using the two copies of a signal registered at a pair of microphones as inputs.

A cross correlation routine outputs set of cross correlation coefficients, which correspond to different time delays. These coefficients are sum of products of corresponding portions of the signals, as one signal slides on top of another one. Each cross correlation coefficient corresponds to a particular possible time delay of a signal between the two microphones. The maximum cross correlation coefficient indicates the portions of the signals with the maximum correlation. Therefore, the maximum coefficient can be used to deduce the time delay between the two copies of the same signal recorded by a pair of microphones. We have used a normalized version of the cross correlation coefficients formula, which is shown in *Equation 1*. The signals that we are cross correlating are x and y. The signals can be represented as row vectors and their

components are referred to by using indices. The mean of these signals is denoted by *m* with subscripts indicative of the signal considered.

$$R_t = \frac{\sum_{l}\sum_{l}(I_x - m_{I_x})(I_y - m_{I_y})}{\sqrt{\left(\sum_{l}\sum_{l}(I_x - m_{I_x})^2\right)\left(\sum_{l}\sum_{l}(I_y - m_{I_y})^2\right)}}$$

Equation 1 Normalized Cross Correlation Coefficients

2.3 The Source Localization Routine

This routine takes the geometry of the problem into consideration. It also uses the cross correlation routine that is described above. The source localization routine is implemented so that it could easily be working on simulated or real data. Regarding the simulation, sound is mixed by performing vector additions in MATLAB, and ignoring complex reverberant features of rooms which bring about the difference between the simulated and real data results. The original commented code for the simulation routine is provided in the *Chapter 4.1*.

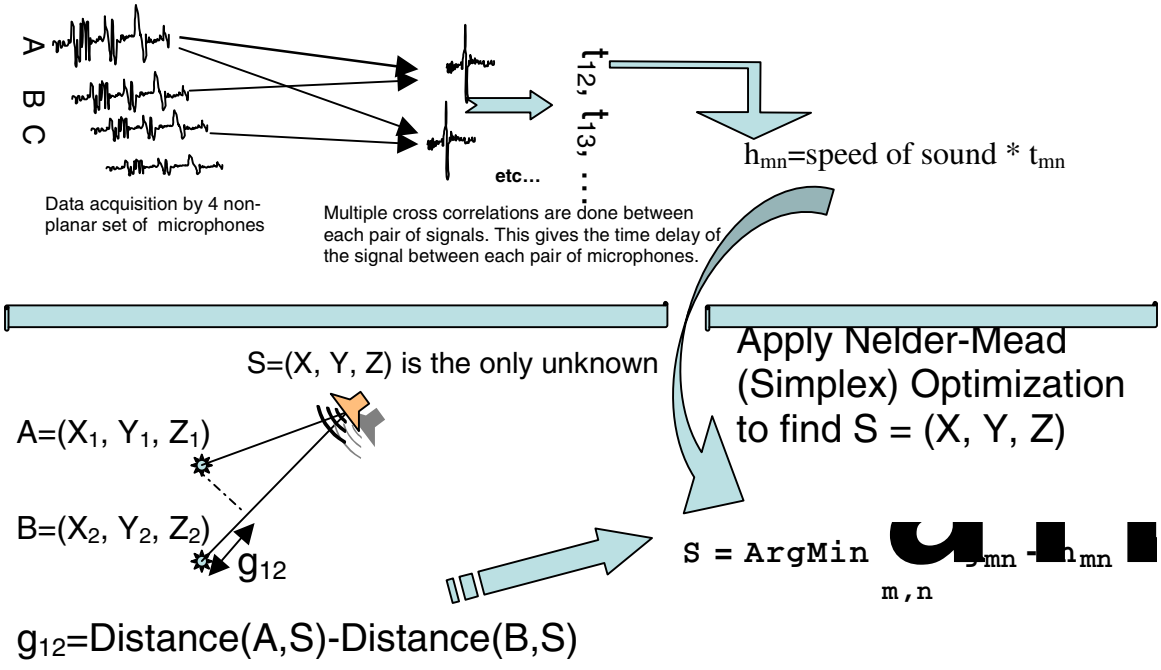

Figure 4 The mechanism of the source localization routine

The source localization routine can take in any number of signals; in our case, it takes in four signals, which are either recorded by a microphone in a lab or simulated by

MATLAB with a predefined geometry inputted into the routine. The routine cross correlates each pair of signals received to generate a set of time delays between each pair of microphones. This time delay information is multiplied by the speed of sound to give information that encapsulates the conceived constellation of the source with respect to the four microphones, whose location is already known.

The goal of the routine is finding the coordinates of the source in a three dimensional space. So far the routine has determined the difference in path length as sound travels from the source to any pairs of microphones. This difference in path length is always expressed in terms of the coordinates of the source that we hope to find and the known coordinates of the microphones. Due to the overdetermined nature of the system and inherent measurement imprecision a value may not exist that defines the variables which represent the coordinates of the source. Therefore, we have to carry out a one dimensional optimization to find the coordinates of the source. The optimization involves minimizing the difference between: the path length difference in terms of the coordinates of the source, and the corresponding number that is found as a result of the cross correlation procedure. This optimization is carried out by applying the standard Simplex optimization scheme in MATLAB; it should be noted that other optimization methods could be implemented. *Figure 4* illustrates the source localization routine described.

2.4 Experimental Setup

The source localization routine was tested by sound recording experiments done in a laboratory. We setup a fixed coordinate system in the laboratory. Four microphones were placed at the tips of an imaginary tetrahedral, whose sides are about 2m long. The microphones were hooked up to a computer, which ran a Lab View program. The program saved four of signals from the microphones into a spreadsheet for later processing by the MATLAB source localization routine. Several sound recording experiments were done by placing a source of sound at various locations in the laboratory.

2.5 Results of Source Localization

The source localization routine was tested on signals simulated for a geometry that consists of microphones placed at tips of a tetrahedral and a source that could be placed anywhere in 3D. The source localization routine turned out being robust with a condition number in order of magnitude of 1, for sources that are close to the center of the tetrahedral. This means a percent error introduced into the locations of the microphones gets amplified by a factor with order of magnitude of 1 before manifesting itself as a percent error in the location of the source determined.

However, we should emphasize that the condition number varied erratically for sources that are placed far away from the center of the tetrahedral. We hypothesize that the robustness is dependent on the region of the source because of the geometry of the problem, which is illustrated in *Figure 3*. As shown in *Figure 3*, the width of the hyperbolic stripes widens as we go further out from the geometrical center of the microphone clusters. A slight misspecification of the location of the microphone makes the wider part of the hyperbolic stripes to sweep a larger area than the area swept by the narrower part of the hyperbolic stripes. Therefore, location of a source far from the center of the microphone clusters is miscalculated as a result of even the slightest error in measurement of the location of the source. The erratic behavior of the condition number of the source localization routine is definitely one of the questions that arose out of this project for future research.

The robustness of the source localization routine, which is region specific, is the justification used to use the minimum number of microphones in our experiments; because the marginal benefit of a fifth microphone is not demanded for most applications, in which the source is known to be in a region that would keep up the robustness of the routine. Basically, we can afford to use only the minimum number of microphones as long as the source is within the imaginary tetrahedral formed out of the microphones.

In addition to simulation testing, the routine was tested on data collected in a lab as explained in *Chapter 2.4*. The condition number of the routine was in order of magnitude of 10 for data that came from the lab. It is hypothesized that the degrading robustness has to do with the reverberation in the lab, which is hard to model while simulating signals. The reverberations can make our cross correlation procedures to yield

inaccurate time delay information. This problem could be ameliorated by using better data acquisition cards and/or performing the localization in an anechoic environment. Addition of a fifth microphone could also be an alternative improvement; however, we were satisfied with the robustness of the system with four microphones.

Chapter 3

Sound Source Separation

3.1 The Geometry of the Problem

Unlike the localization techniques used, our source separation techniques rely on differences in the intensity of sound signals recorded by microphones placed at different distances from the source. The intensity of sound signal is known to be proportional to the inverse square of the distance from the source of sound. However, this relationship may not be observed in a signal recorded in a closed space because of the reverberations and the interferences that result. This is particularly the case when there is a periodic pattern within the signals.

In this project, we only deal with the extraction of two sources from two mixtures that they formed. Therefore, the sources have to be constrained to a certain geometric region so that the two mixtures are distinct. This constraint can be lifted very easily by adding additional microphones; therefore, the source separation that we do is still blind, but computationally simpler than the usual problem encountered.

The loci of points of aliases that arise in this problem are different from the ones discussed in *Chapter 2.1*; because Independent Component Analysis relies on intensity difference information. In the case of two microphones and two sources in 3D, we would have to make sure that we do not get the same exact mixtures at the two microphones. Therefore, the sources should not be placed on the plane that perpendicularly bisects the line segment that connects the two microphones; because it is the locus of points of aliases for our system.

3.2 Projection Pursuit

Projection Pursuit is a source separation technique, which is reminiscent of Independent Component Analysis (ICA) that is used in this project. It is advantageous to study Projection Pursuit; because it is less complicated than ICA. On the other hand, Projection Pursuit is a serial method, which makes it inefficient for real applications. On the other hand, ICA takes sets of mixtures of sources as an input and returns sets of estimates of the sources at once, i.e. it is a parallel technique.

3.2.1 Modeling the Mixing Process

While modeling the sound mixing process, we develop notation thatis used consistently throughout the discussion of sound source separation using Independent Component Analysis. Sampled sound sources are usually represented as column vectors. Hence, we can represent the mixing process by defining a mixing matrix as follows.

[s1, s2] * M = [m1, m2]

Equation 2 The Mixing Model

Here s1 and s2 stand for source 1 and source 2, whereas m1 and m2 are mixed signal 1 and 2 that are recorded by the two microphones. The mixing matrix is two by two. In ideal situations, the mixing matrix depends on the geometry of the problem. The amplitude of sound signals is proportional to the inverse of the distance from the source to the point of recording. Therefore, the inverse of each component of the mixing matrix is proportional to the distance between each pairing of source and microphone possible. Once again, this is not quitetrue in reality because of the effect of interference and reverberation.

3.2.2 Unmixing

We are interested in extracting the sources from the mixtures. So, we extend the mixing model further and express the estimates of the source signals as a product of the unmixing matrix, which is the inverse of the mixing matrix, and the recorded signals. In *Equation 3*, y1 and y2 refer to the estimates of s1 and s2 respectively.

[y1, y2] = [m1, m2] * U

Equation 3 The Unmixing Model

Evidently, retrieving y1 and y2 would be trivial if M, and hence U, was known. But, the mixing process is not known as we are working on a blind source separation problem. So, we can try different values of U (inverse of M) and pick the best estimate of the sources which results. The problem is determining which estimate of the source is a good one; because we do not know what the source is.

If we assume that all the sources are sampled from the same probability density function, we can stop adjusting the components of U when the estimates of the sources have a probability density function that matches our assumption. If we are working on speech source signals, it is reasonable to assume that the sources are sampled from a

super-Gaussian probability density function; because speech signals are characterized by a sequence of pauses in between words that result in a probability density function that peaks at zero.

Kurtosis is the fourth statistical moment which measures how peaky the probability density function of a signal is. The kurtosis of a Gaussian signal is zero and sub-Gaussian signals have a negative kurtosis, whereas, super-Gaussian signals are characterized by a positive kurtosis. In light of this, projection pursuit adjusts the components of U until the kurtosis of y1 and y2 are maximized. Analytic expression of the kurtosis of the estimates in terms of the components of U proves helpful in expediting the optimization needed to find U. [5] Optimization methods such as gradient ascent are used to find a value of U that maximizes the kurtosis of the estimates.

3.3 Independent Component Analysis (ICA)

Independent Component Analysis can be considered as the extension of Projection Pursuit. It also makes assumptions about the probability density function of the source signals. Furthermore, it assumes that the source signals are statistically independent. Overall, ICA is capable of extracting sources that are independent and identically distributed, as long as it knows the probability density function of the i.i.d. signals.

Independent Component Analysis, which is applied in this project, is the work of Bell and Sejnowski, who published a revolutionary article in the field of blind signal separation in 1995. [4] Their article has been eloquently rephrased in an introductory book written by James Stone. [5] The author of the book included a code that implements ICA. We customized his MATLAB code, which is included in *Chapter 4.2*.

3.3.1 The Mechanism of ICA

ICA adjusts the components of U in order to make y1 and y2 as mutually independent as possible; because it is assumed that source signals are statistically independent of each other. Whereas, mixtures of source signals are not independent of each other as they are made out of the same underlying sources despite the different intensity of the constituents.

Bell and Sejnowski have suggested an information maximization approach in order to produce estimates of sources that are mutually independent of each other. [4] ICA assumes the probability and hence cumulative density function of the source signals; therefore, we can pass our preliminary estimates of the sources (y1 and y2) through the cumulative density function of the sources to get signals Y1 and Y2. Bell and Sejnowski have shown [4] that y1 and y2 become mutually independent if we can choose an unmixing matrix, U, which can maximize the joint entropy of Y1 and Y2. Furthermore, Bell and Sejnowski have expressed the joint entropy of Y1 and Y2 in terms of the components of U. Therefore, ICA maximizes the joint entropy of Y1 and Y2 by adjusting the value of U systematically. This results in estimates y1 and y2, which are as mutually independent as possible.

3.3.2 Simulations to test ICA

We have customized James Stone's ICA code and have gotten results that are shown in *Figure 5*. We can see that ICA reproduces the source signals with amplitude and order indeterminacy.

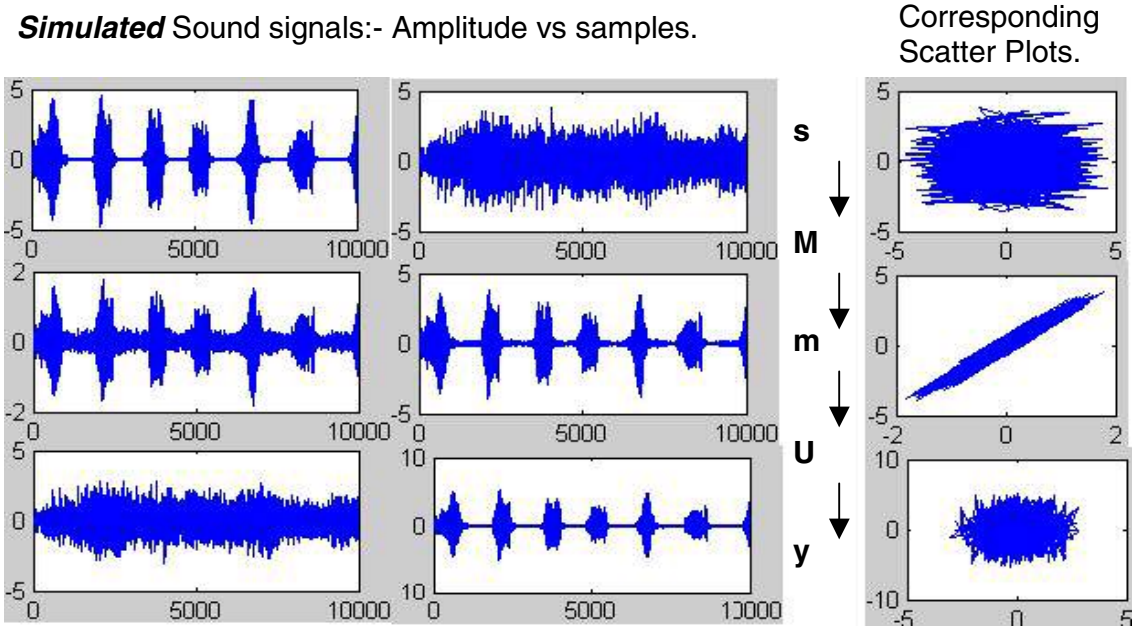

Figure 5 Simulation done to test ICA. The six graphs to the left show the signals: Intensity vs. time /ms. The scatter plots to the right show the corresponding three pairs of signals to the left.

3.4 Limitations of ICA

Despite ICA's success in separation of other kinds of signals (such as electrocardiogram applications, image separation), it is problematic when applied to audio separation. The major problem is that the mixing model that ICA relies on does not consider the time delays that are introduced between copies of signals recorded by different microphones. These time delays are very small compared to the sampling period of most data acquisition systems for signals that travel fast, such as light. However, the time delays that arise in this situation are not always smaller than the sampling period of our data acquisition system. Therefore, the mixing matrix can not be a sufficient way of modeling the mixing process. Therefore, ICA can not be directly applied to extract sources from mixtures collected by microphones that are placed wide apart in space.

As we mentioned before, ICA relies on intensity difference information as opposed to time difference information. However, the intensity difference information is prone to distortion by interferences in a reverberant room. Once again, the mixing model that was discussed assumes an anechoic sound mixing environment. Hence, our results are expected to be negatively affected by reverberations in the environment.

3.5 Traditional Extensions of ICA

3.5.1. Torkolla's Delayed and Convolved Extension of ICA

This technique is designed to solve the delay problem that is outlined in the previous section. It implements the optimization of delay using a feedback neural network. We have implemented this extension of ICA solely from the description of the algorithm in the article published by Torkolla [2]. However, our results could not replicate the results that were reported in the article, hence, further progress in applying this technique was not possible.

3.5.2. Lambert's FIR matrix algebra representation of sound mixing

This uses Finite Impulse Response filters, as opposed to numbers, as elements of the mixing matrix. This enables us to represent not only delays but also to model reverberant atmospheres. The FIR are basically implemented as vectors. Russel Lambert wrote his PhD thesis [3] on the applicability of such a representation for blind source

separation. The attempt to use this representation to implement ICA has not been successful.

3.6 ICA Using Closely Spaced Uni-Directional Microphones

3.6.1 Overcoming the Limitation of ICA

In *Chapter 3.4* we discussed the limitations of ICA when it is applied to audio signals. The first problem was that the time delay between copies of signals recorded at different microphones is not significantly smaller than the sampling period of our data acquisition system. This is a problem for audio separation because of the relatively low speed of sound. The conditions that are necessary for the success of ICA in signal separation are illustrated in *Figure 6*. Even though the speed of sound is small and invariant, we can physically minimize the gap between the microphones in order to minimize the time delay. If we place the microphones almost adjacent to each other with a gap of about 1cm, the time delay introduced is less than one tenth of the maximum sampling period required for signals with frequency that is less than 1KHz. The maximum sampling period is determined by the Nyquist Criteria, which states that the minimum sampling frequency is twice the maximum frequency component of the signal.

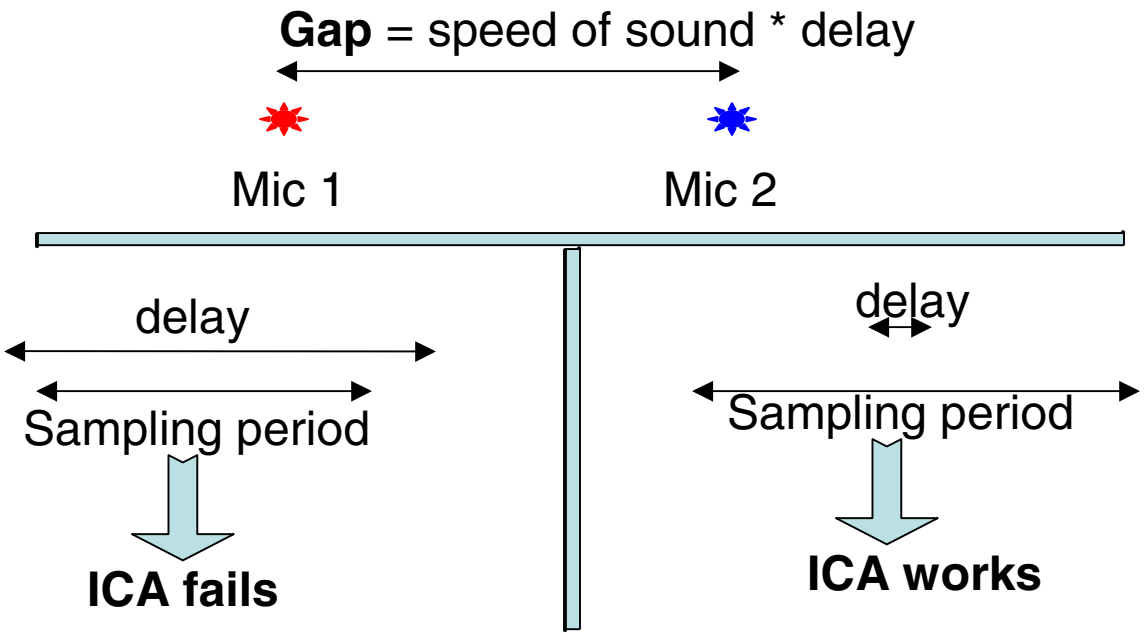

Figure 6 Conditions of the delay problem

Placing the microphones close to each other seems to solve the delay problem. However, it is at the cost of losing the intensity difference information that ICA relies on. Because, closely spaced omni directional microphones record exactly the same copy of signals. In other words, the microphones would be redundant. As a result we suggest the use of Closely Spaced Unidirectional Microphones which are aligned in different directions. Their physical orientation allows them to pick up different copies of the signals.

3.6.2 Unidirectional Microphones

These are available in the market, and have a limited frequency band. We have unidirectional microphones with sensitivity from 100Hz to 1000Hz. The basic physical mechanism of these microphones is depicted in *Figure 7*. Sound that arrives at the back of the microphone hits the diaphragm on both sides at about the same time; therefore, it fails to register a pressure gradient on the diaphragm. On the other hand, the physical structure of the microphones prevents this from happening for sounds that come from any other directions, particularly from the front of the microphone. The polar pattern of directional microphones is in general frequency dependent. However, their manufacturers mitigate this problem by using complicated acoustic foams and holes, whose workings are not entirely clear. This results in a cardioid polar pattern that indicates low sensitivity at the back of the microphone.

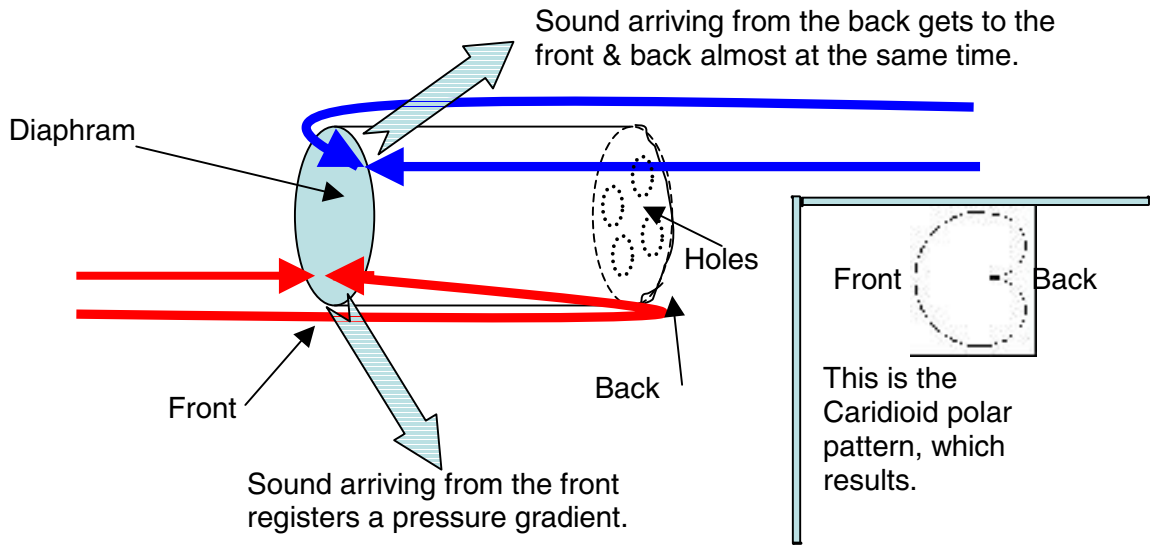

Figure 7 The basic physical mechanism of Unidirectional Microphones

We have performed a polar pattern characterization experiment on our unidirectional microphones. Here is a description of the set up of the experiment. A monotone sound source with adjustable frequency was set up on a fixed stand 10 cm away from a unidirectional microphone set up on a stand with a degree of freedom for rotation. The back of the unidirectional microphone was oriented at certain angles from 0 to 180 degrees with respect to the line joining the source with the microphone. At each angle of orientation of the microphone, the source played monotones at 10 frequencies ranging from 100 to 1000 Hz. The result of the experiment is depicted in *Figure 8*. As can be seen on the graph, the polar pattern of the unidirectional microphones turned out to be frequency independent for this frequency band.

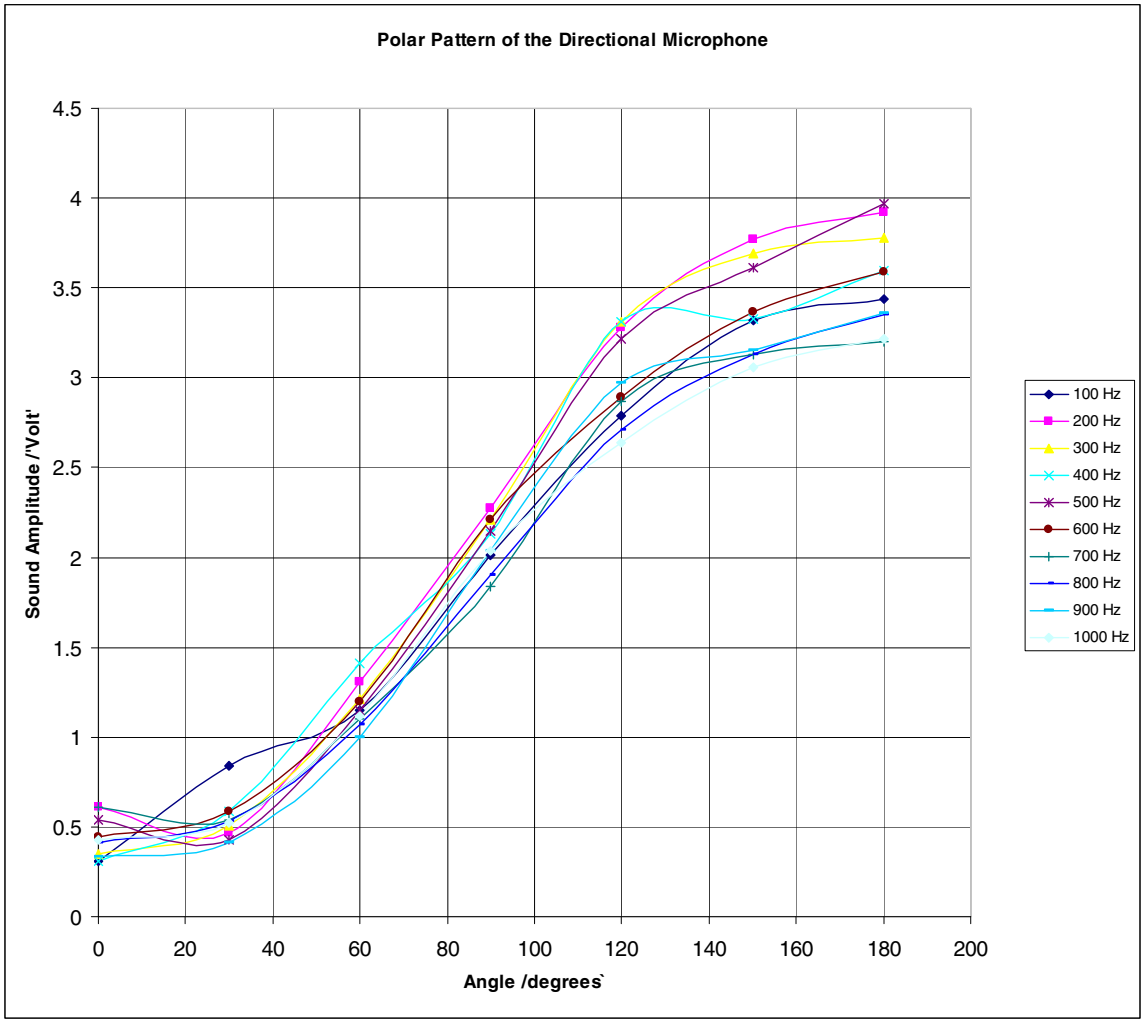

Figure 8 Characterization of the polar pattern of unidirectional microphones

3.6.3 The New Geometry of the Problem

Consider the case where we have two closely spaced unidirectional microphones that are facing in opposite directions. In this situation, there are several source constellations, which should be restricted in order to record different copies of the mixture of source signals. For instance, the plane that perpendicularly bisects the cluster of these microphones, so that the front and back of each of the microphones is on opposite sides of the plane, makes the microphones redundant. Besides, this system can not separate sources that are on a line that also passes through the center of the unidirectional microphone cluster. Therefore, the sources should be placed in a way that allows us to record different copies of the mixture of the sources using the two microphones.

The microphone cluster set up discussed in the previous paragraph has the following shortcomings if each of the two sources are placed facing the front of each of the microphones. Because each of the recorded mixtures become dominated by a single source. Evidently, this makes source separation a trivial exercise as the mixtures themselves are good approximations to the sources. Therefore, we refrain from oversimplifying the problem by avoiding this configuration.

An alternative configuration which provides us with sufficient source separation challenge while allowing us to record mixtures of signals that ICA can separate is described below. The unidirectional microphones are placed so that the lines that go through them from back to front form ninety degree with respect to each other. In this configuration, a source faces the front of one unidirectional microphone and the side of another one. This results in mixtures with fair proportions of the individual sources; thus, the source separation is not trivial as would be in the configuration discussed beforehand.

3.6.4 Results

This technique has proved to be a successful application of ICA by separating mixtures of two low frequency narrow band signals (a person counting 1 to 3 & ‘an alarm clock’). The use of better unidirectional microphones, with a broader band, is expected to give a better result. The two mixtures and the estimates of the narrow band non-Gaussian sources is illustrated in *Figure 9*.

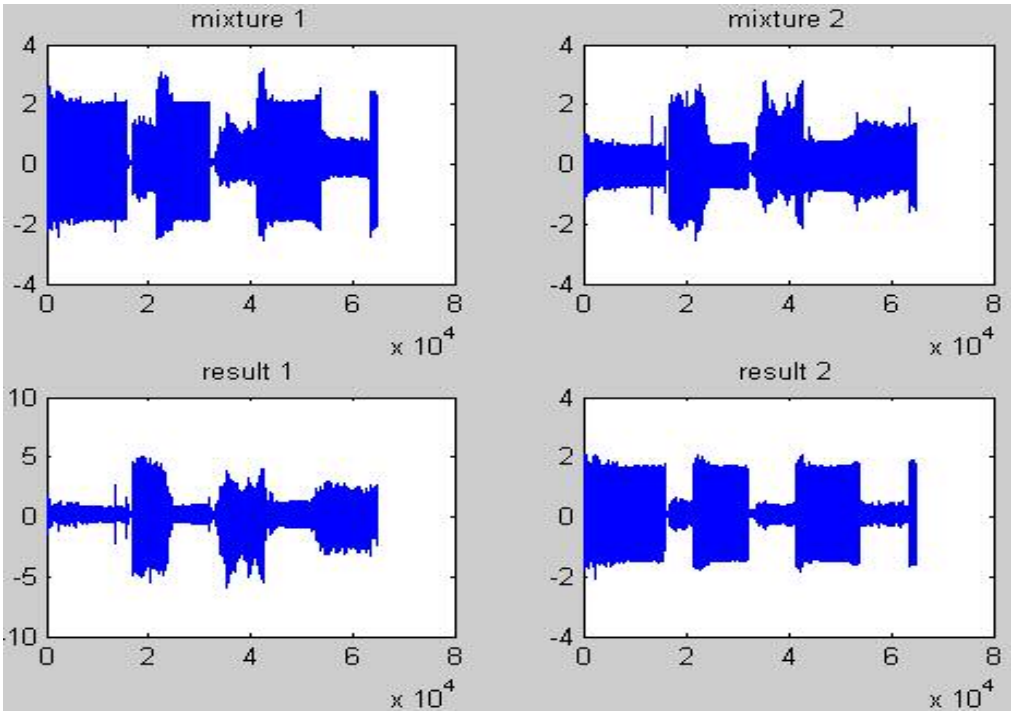

Figure 9 Results of source separation using Closely Spaced Unidirectional Microphones. The graphs show sound signals: Intensity vs. time in units of 0.1ms.

3.6.5 Evaluating the Results

The choice of the source signals for a plausible experiment was dependent upon two factors: the probability density function of the sources, and the frequency make up of the sources. The unidirectional microphones are frequency independent only for the range of frequencies from 100Hz up to 1KHz. Therefore, we had to use a signal generator to produce an alarm clock sound with the desired frequency range, and we used a low frequency speech. The alarm clock sound was chosen to have a similar probability density function as that of speech. However, the probability density function of the alarm clock sound has two more peaks in addition to the characteristic super-Gaussian peak at zero. Therefore, ICA's assumption regarding the probability density function of the sources is not entirely valid in this case. Nevertheless, ICA was able to extract an estimate of the source signals as shown in *Figure 9*.

In light of this observed robustness of ICA, we assessed the sensitivity of ICA to misspecification of the probability density function of the sources. We took advantage of the customized ICA code, included in *Chapter 4.2*, to generate source signals that have a

probability density function similar to that of an alarm clock sound. Once again, such a probability density function has two more peaks than the zero peak that is characteristic of a super-Gaussian signal; from now on, we shall call such a signal a '*pseudo super-Gaussian signal'*. We used the customized ICA code to mix two different pseudo super-Gaussian narrow band signals illustrated at the top left corner of *Figure 10*. The sound signals are mixed with a small time delay, which would arise in a sound recording experiment using closely spaced unidirectional microphones. The signals and their mixtures as well as the corresponding probability density functions are shown in *Figure 10*. The estimates of the source signals computed by ICA are also illustrated in the same Figure.

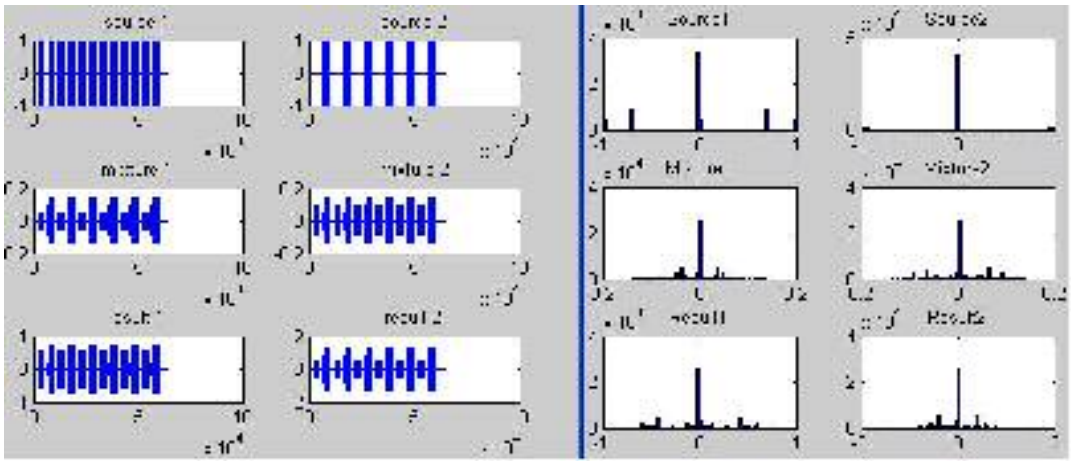

Figure 10 Generating, mixing and separating 'pseudo super-Gaussian' narrow band signals. The six graphs to the left show signals: Intensity vs. time in units of 0.1ms. The graphs to the right show the corresponding probability density functions of the signals to the left.

Qualitatively, the estimates of the sources by ICA are not significantly different from the mixtures as shown in *Figure 10*. This is because ICA assumed that the source signals are super-Gaussian with a cumulative density function of hyperbolic tangent, while the sources are actually only 'pseudo super-Gaussian' as shown in the top right corner of *Figure 10.*

We need to compare the result of this simulation experiment with a similar experiment where ICA's assumption is more reasonable. For the second sound mixing and separating simulation, we modify the signals used in the previous experiment so that they have a super-Gaussian probability density function; then, we mix them in MATLAB. We shall discuss the technique used to transform a signal into its own variety

with a specified probability density function first before comparing and contrasting the results of the two sound mixing and separating simulations.

Chapter 4.3.2 contains a program, which takes in two signals and transforms the first one to its own variety with a probability density function of the second model signal. This code was provided by Professor Kaplan. Our objective is to transform the pseudo super-Gaussian source signals to their super-Gaussian version. This can be achieved by the program in *Chapter 4.3.2*. The only missing part is a model super-Gaussian signal with a cumulative density function of hyperbolic tangent, which is consistent with ICA's assumption. I have implemented the "Rejection / Acceptance method" to generate a model signal with a hyperbolic tangent c.d.f.; the MATLAB implementation is included in *Chapter 4.3.3*. Now, we shall discuss some of the details of the Rejection / Acceptance method.

Rejection / Acceptance Method:-

This method [6] relies on choosing a Comparison Function, which is above the target pdf throughout the domain of both functions. Besides, the Comparison Function should have an analytic integral and an inverse for this integral. The Bell Sejnowski ICA assumes that source signals have a cdf characterized by the Hyperbolic Tangent function. Therefore, the target pdf is the square of a Hyperbolic Secant function. The comparison function in *Equation 4* is used to simulate a Gamma distribution, and it can also be used for our desired distribution, suitably.

```
f(x) = 1.5 / (1 + (x/2)^2)
```

Equation 4 Comparison function used in the Rejection / Acceptance method.

Look at *Chapter 4.3.3* for more details regarding the implementation of the rejection / acceptance method. The method has been tested and worked as desired. *Figure 12* illustrates the way one of my simulated pseudo super-Gaussian monotone signals had been modified to fit the Bell Sejnowski assumptions regarding the pdf of a signal.

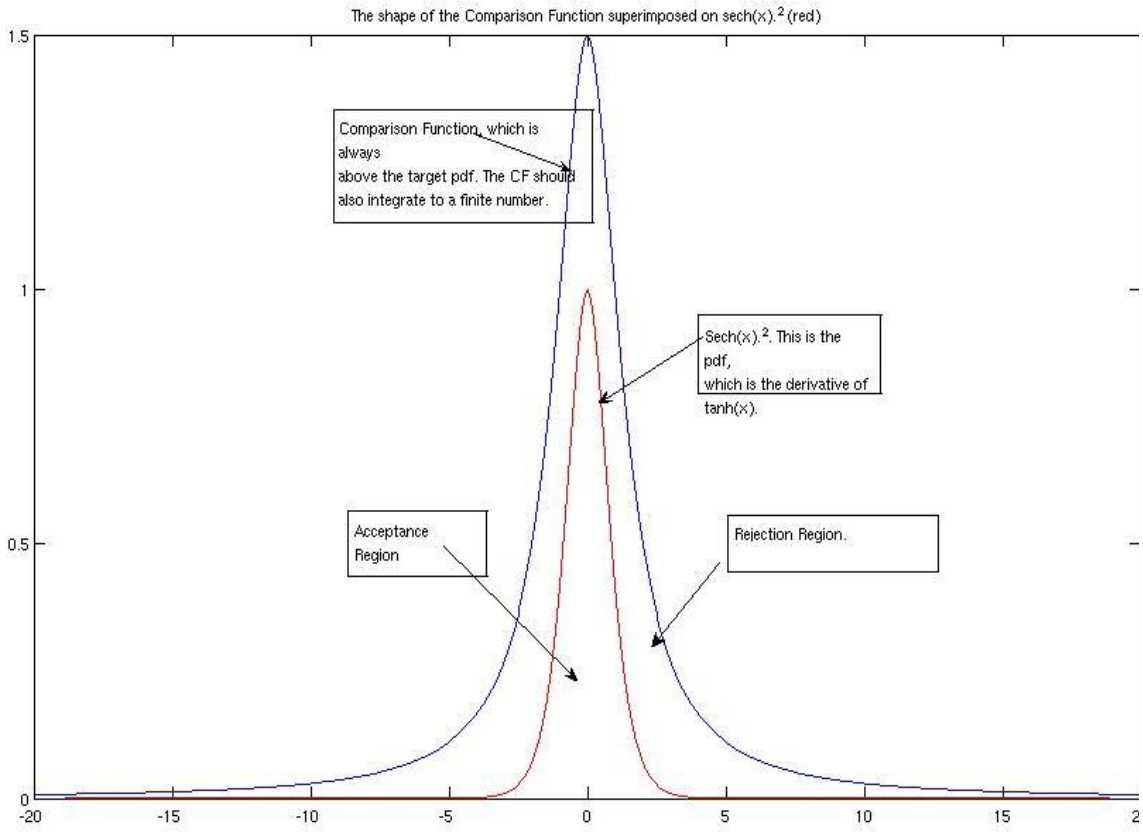

Figure 11 This illustrates the rejection/acceptance method. Look at *Chapter 4.3.3* for more details.

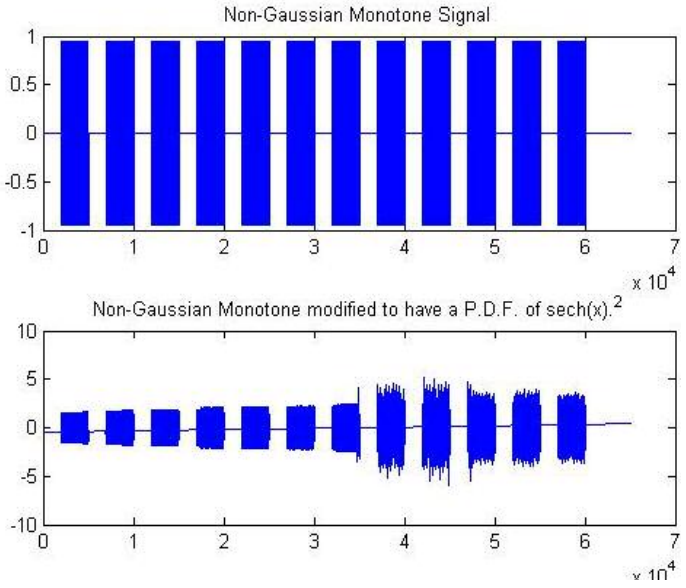

Figure 12 Testing the Rejection/Acceptance Implementation. The graphs show signals: Intensity vs time in units of 0.1ms.

Having discussed the mechanism of generating a super-Gaussian version of a pseudo super-Gaussian signal, we can resume our discussion of the sound mixing and separating simulation performed. The super-Gaussian versions of our original signals have been mixed and separated using ICA in a similar manner as the signals illustrated in *Figure 10*. The signals with a modified pdf are used in the second experiment. The results of the second experiment are illustrated in *Figure 13*.

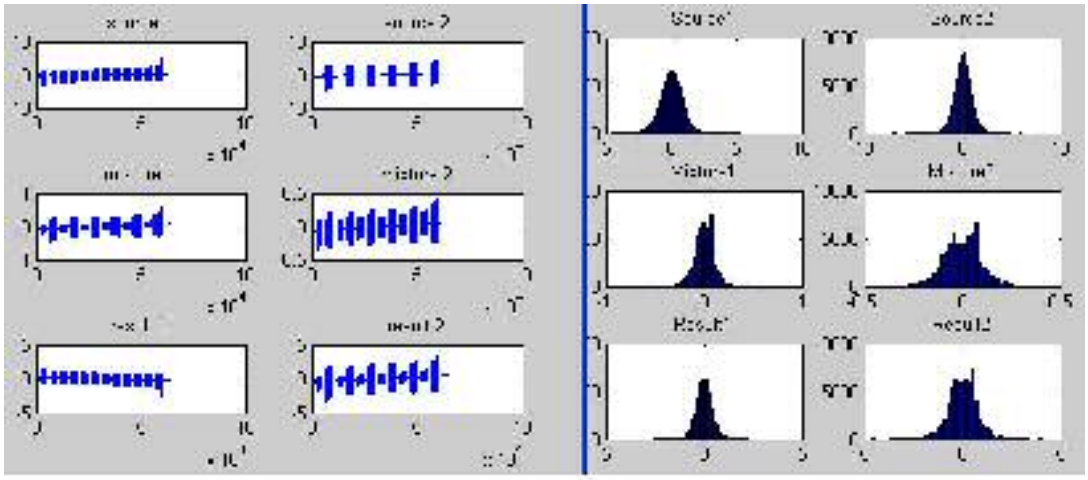

Figure 13 Mixing and separating super-Gaussian (after modification of previously pseudo super-Gaussian signals) narrow band signals. The six graphs to the left show signals: Intensity vs. time in units of 0.1ms. The graphs to the right show the corresponding probability density functions of the signals to the left.

Qualitatively, we can compare and contrast the success of the two experiments by looking at *Figure 10* and *Figure 13*. Apparently, the results in *Figure 13* are better estimates of the source signals than the ones in *Figure 10*. This is because of the invalidity of the assumption thatICA made regarding the probability density function of the sources. Under normal circumstances, quantization of ICA's success is demanding because we actually do not know the sources that we think we are estimating. However, in this case it is possible to quantize the success by measuring the correlation between the sources and estimates of our simulation experiment.

Specifically, we measure the correlation between each source-estimate pairing possible. We have two signals and their two estimates with order and amplitude indeterminacy. Therefore, we have four correlation coefficients, which are presented in the form of a two by two matrix. The correlation coefficients are returned by the program in *Chapter 4.2* when it runs such sound mixing and separating simulations. The following

coefficients were returned, when ICA made a reasonable assumption (i.e. in the second experiment).

0.9973 0.5804
0.1389 0.8510

Under ideal separation circumstances, two of the above four coefficients would be 1 and the others would be zero. One of the estimates is definitely far from being an ideal estimate. However, other estimate is definitely an apt approximation to the source. Overall, we have proved that ICA works reasonably well to separate mixtures that would come from closely spaced unidirectional microphones.

In the first experiment, we had mixed two pseudo super-Gaussian signals before separating them. The quality of the separation is measured by the following coefficients.

0.9807 0.6508
0.1956 0.7593

We can see that the quality of the separation degrades as the probability density of the sources deviates from ICA's assumption.

Therefore, we can conclude this section by noting that the idea of using closely spaced unidirectional microphones to implement ICA is valid. However, the frequency of the sources needs to be low enough for the unavoidable time delay between the closely spaced unidirectional microphones to be immaterial to themixing model that ICA relies on.

3.7 Source Separation Using Time Delay Information

In this part, we would like to simplify the problem by using only 2 microphones to separate a mixture of 2 sound sources. We use time delay information as opposed to intensity difference information used in ICA. Therefore, we should take the geometry of the problem discussed in *Chapter 2.1* into consideration while setting up the experiment. For example, we won't be able to separate mixtures of sources if both of the sources are placed on one of the hyperbolic curves shown in *Figure 2*; because there is no meaningful time delay information that we can extract by taking the cross correlations of the signals.

For simplicity we use a configuration that is guaranteed to avoid such 'alias problems' discussed in the previous paragraph. The sound sources are placed on the opposite ends of the line, which perpendicularly bisects the segment that connects the

microphones. So, the two delay values that we usually get out of the cross correlation function have opposite signs This is an artifact of the implementation details of our Labview program.

3.7.1 Modeling the mixing process*:-

Here we establish notation, and a model of the mixing process based on our setup of sources and microphones.

S1 & S2 are vectors representing the two sound Sources which are played in the lab.

M1 & M2 are the two Mixtures of these sources recorded by the 2 microphones.

X1 & X2 are approximations of S1 & S2, which we are trying to generate.

According to our experimental setup described above, the following model describes the sound mixing process.

M1 = S1 + a*S2(d)

M2 = a*S1(-d') + S2

The two sources are placed close to the two microphones. So, M1 picks up S1 as it is. But, S2 gets delayed by an amount "*d*" and it is also attenuated by a factor "*a*" before it contributes to forming M1. The same argument describes the second equation above. Note that the delay values have opposite signs; because, the sound sources are placed on the opposite sides of the microphones.

According to this model, we only need to find the values of *d* and *d'* in order to generate an approximation of S1 and S2 from M1 and M2. The value of the attenuation factor is irrelevant as we will see later.

3.7.2 Cross correlating the mixtures:-

We expect to see two distinct peaks, which correspond to the values *d* and *d'*, in the cross correlation of M1 and M2. We perform a 'normalized' cross correlation in order to enhance the visibility of these two peaks, i.e. *Equation 1* is employed. We consider the two highest peaks of the cross correlation of M1 and M2 to deduce values for *d* and *d'*. These delay values are then used to shift M1 and M2 appropriately before adding them together to generate X1 and X2. Such a procedure is commonly known as the delay and sum technique. [7]

* This is the only section of Chapter 3, where we introduce a different mixing model.

3.7.3 Generating X1 and X2:-

X1 is the sum of a version of M1 (which is padded with *d* zeros at the beginning) & M2. X2 is the sum of a version of M2 (which is padded with *d'* zeros at the beginning) & M1. Performing this addition should intensify one of the source signals. The other source signal stays there, but, it can be considered as a background noise when compared to the intensified signal. The generated signals X1 and X2 are displayed on a Lab View interactive program along with the recorded mixtures and the cross correlation graph.

3.7.4 Conclusion:-

The biggest setback to this approach of separating mixtures of sounds is the difficulty of finding multiple peaks of the cross correlation function. Usually, there are false peaks, whose origins are not understood yet. Besides, the amplitude of the two sound sources should be comparable. Otherwise, the peak that would correspond to the delay-information of the quieter sound source would be hidden in the noisy cross correlation function.

It has been possible to identify two reasonable peaks at times when the amplitudes of the two sources have been adjusted very well in order to match each other. In such cases, it is possible to observe a crude approximation of S1 and S2 through X1 and X2. The performance of the system (assuming the existence of a reliable cross correlation peak identification routine) could be improved by adding more microphones and hence considering more cross correlations.

Chapter 4

Appendix

4.1 Source Localization Routine in MATLAB

This is a MATLAB routine that I wrote to implement the source localization routine described in Chapter 2.

```
function [answer, condition] = localize1(simu, source, dataFile)
% Returns the estimated location of a single source from the delay
% information calculated using the cross correlation function AND
% the condition number of this estimation.
% Works on any number of microphones used to collect the data.
% The locations of the microphones should be hardcoded.

% simu -> simulation? :: 1=simulation 0=real data
% source -> location [x, y, z]. This is crucial during simulation.
% dataFile -> EXCEL file saved by the LabView data acquisition program
% UNITS: Distance:- meters     Time:- seconds.


c = 340;    % Speed of sound

% Microphone Positions. This is experiment specific!
% [x1 y1 z1;
%  x2 y2 z2;
%  x3 y3 z3; ...
%  xn yn zn]
%%% MODIFY AS NEEDED
% MicLoc = [0.0, 0.0, 0.0; 1.0, 0.0, 0.0; 0.0, 1.0, 0.0; 1.0, 1.0,
1.0];
MicLoc = [
         262, 0, 0;
         85, -145, 0;
         0, 0, 0;
         0, -145, -74
         ];
% The following is an off value for the microphone locations. It helps
% us to estimate the condition number of the routine during
simulations.
OffMicLoc =[
         262.5, 0.5, 0.5;
         85, -145, 0;
         0, 0, 0;
         0, -145, -74
         ];
% Make a good guess for the location of the source.
Guess = mean(MicLoc, 1);


%%% Comment out ONE OF THESE TWO as necessary.
```

```
% *1.   Work on real data.
if (simu==0)
    delays = generateDelays(dataFile, size(MicLoc,1))

% *2.   Work on simulated data for testing purposes.
% Specify the location of the source to get
% the delays in this microphone configuration.
elseif (simu==1)
    %%%PUT SOURCE HERE for SIMULATION
    delays = simDelay(MicLoc, source);%%%MODIFY AS NEEDED
end

%%% Comment out ONE OF THESE TWO as necessary.
% If the system is NOT overdetermined, we have two ways of solving it.
% *1. Solve a system of non-linear equations using Newton's method.

% % % answer = newtonMethod('ellipseFun', Guess, MicLoc, delays);

% *2. The above approach may fail if there are more than 3 microphones,
due
% to overdetermination. Here is a more general approach.
% Find the ArgMin of this function.
global BUSHMicLoc;
global BUSHDelays;
if (simu == 0)
    BUSHMicLoc = MicLoc; % If this is not a simulation, use the values.
elseif (simu == 1)
    % During simulation, use microphone values, which are slightly off.
    BUSHMicLoc = OffMicLoc;
end
BUSHDelays = delays;
answer = fminsearch(@ellipseMerit, Guess);


%%% ASSESS the accuracy of the answer. (we should know where the source
is)
forwardError = (norm(answer - source))./(norm(source));
% estimate the backward error, which describes how accurately we know
% the location of the microphone & the delay. (sometimes, the accuracy
% of the speed of sound can be a factor too.)

% During real experiments Microphone locations are known to within 2cm.
% During simulation the uncertainty introduced should be entered in
here.
if (simu == 0)
    backwardError = (2)./100;
elseif (simu == 1)
    backwardError = (norm(MicLoc-OffMicLoc))./(norm(MicLoc));
end

% Evaluate the condition number of the problem.
condition = forwardError./backwardError;
%%display(condition);
```

```
%%% SUGGESTED TESTING (based on simulation):
% % % %Plot Merit vs z, constraining x&y to the exact souce coordinate.
% % % %EXPECT to see a minima, where z coincides with the source z-
coordinate!
meritVal = [];
for zValue = ((-
10.*abs(source(3)))+source(3)):((10.*abs(source(3)))+source(3))
    meritVal(end+1) = ellipseMerit([source(1), source(2), zValue]);
end
zValues = ((-
10.*abs(source(3)))+source(3)):((10.*abs(source(3)))+source(3));
figure(2); plot(zValues, meritVal); title('TEST: Merit vs z');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function answer = generateDelays(dataFile, nMic)
% This automates the use of crossCorrN to find delays
% between pairs of signals.
% It returns an array of delays.

% dataFile -> EXCEL file by the LabView data acquisition program.
% nMic -> number of microphones used

data = load(dataFile);

delays = [];
for k = 1:nMic
    for j = k+1:nMic
        delays(end+1) = crossCorrN( data(:,k), data(:,j) );
    end
end
answer = delays;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function answer = simDelay(mPos, sPos)
% It computes the delay b/n signals recorded at different microphones.
% It takes locations of microphones and the source.
% sPos -> [x y z]
% mPos -> [x1 y1 z1;
%          x2 y2 z2;
%          ... ;
%          xn yn zn]
% Works for any number of microphnes and a single source.

% Speed of sound
c = 300;

nMics = size(mPos, 1);
delays = [];
%micPairs = []; We probably don't need this SUGGESTED labeling.

% Get the delay b/n all pairs of microphones.
for k = 1:nMics
```

```
    disToK = distance(sPos, mPos(k,:));
    for j = k+1: nMics
        disToJ = distance(sPos, mPos(j,:));
        delays(end+1) = (disToK - disToJ)./c;
    end
end

% Return the delays
answer = delays;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function answer = crossCorrN(x, y, sampleRate, nSamples)
% It cross correlates the two signal parameters to find thier delay.
% It does a normalized cross correlation. This avoids ambiguities in
% picking a maximum.
% It makes advantage of reality (the fact that the delay can't possibly
% be bigger than *0.2* second in a room) to hasten the program.

% x & y -> signals to be cross-correlated.
% sampleRate -> sample rate of data acquisition.
% nSamples -> number of samples in the data acquisition.

%%%% SET these appropriately!!!
if nargin == 2   % Default Data Acquisition values
    sampleRate = 15000;
    nSamples = 65000;
end

minD = nSamples-floor((0.1.*sampleRate)); % 0.1. B/c max(Delay)=*0.2*
here
maxD = nSamples+floor((0.1.*sampleRate));

%%lengthR = maxD - minD +1;
lengthR = length(x) + length(y) - 1;

% % % minD = 1;
% % % maxD = lengthR;
R = zeros(1, lengthR);

% Do a normalized cross correlation.

meanX = mean(x);
meanY = mean(y);
denominatorX = sum((x - meanX).^2);
denominatorY = sum((y - meanY).^2);


denominator = sqrt(denominatorX.*denominatorY);

% The following is the common 'sliding business' of cross correlation.
for t=minD:maxD
    numerator = 0;
    for i =1:length(x)
        p = t - length(x);
        if ((i+p <=length(y)) & (i+p>=1) )
```

```
            numerator = numerator + ((x(i)-meanX).*( y(i+p) -meanY));
        elseif (i+p<1)
            % Zero padding
            numerator = numerator + (x(i)-meanX).*( 0 - meanY);
        else
            numerator = numerator + ((x(i)-meanX).*( 0 -meanY));
        end
    end
    R(t) = numerator./denominator;
end

% Find the delay
[maxim, argmaxim] = max(R);
answer = nSamples - argmaxim;   % +ve -> D(x)>D(y) & -ve -> D(x)<D(y).
% Plot the cross correlation coefficients. (for verification by eye)
figure(1); plot(R,'.'); title('Cross-CorrelationN coeeficients');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function answer = distance(a, b)
% implements the Euclidean distance formula to find
% the distance b/n the two points given.

if(length(a)~=length(b))
    error('Dimensions do not match!');
end
answer = sqrt(sum((a-b).^2));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function answer = ellipseMerit(s)
% Defines a function to be minimized so as to localize a source.
% s -> sound source position.   [x y z]
% m -> microphone positions. (known: in the reverse problem)
% [x1 y1 z1;
%  x2 y2 z2;
%  x3 y3 z3;
%  x4 y4 z4]
% Delays -> d12, d13, d14, d23, d24, d34

% Find a vector of differences b/n reported delays and (geometrically)
% expected delays. Then, sum the squares of the vector.
% Because, we are aiming to minimize the sum of the squares in the
over-
% determined case of more microphones than 3 (b/c we live in 3D).

global BUSHMicLoc;
global BUSHDelays;

vec = ellipseFun(s, BUSHMicLoc, BUSHDelays);
% In the future, divide each element by the PRECISION of the delay
% estimation. This could be the full width at half maximum of our
% cross correlation coeefficients graph.
answer = sum(vec.^2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function res = ellipseFun(s, m, delays)
% s -> sound source position.   [x y z]
% m -> microphone positions. (known)
% [x1 y1 z1;
```

```
%  x2 y2 z2;
%  x3 y3 z3;
%  x4 y4 z4]
% Delays -> d12, d13, d14, ... d23, d24,... d34...

nMic = size(m, 1);
% This is a clever way of calculating the # of Pairs of microphones.
nPairs = (nMic.^2 - nMic)./2;

% So, we will have as many equations as pairs of microphones.
res = zeros(nPairs,1);

% speed of sound
% Distance:- meters     Time:- seconds
c = 300;

% Equations
% We expect the delay reported by cross correlation to equal
% the difference between the distances of the source to each microphone
% divided by the speed of sound.

% Distances b/n microphones and the source.
Dis = zeros(nPairs,1);

h = 1;
for k = 1:nMic
    for j = k+1:nMic
        diffD = distance(s, m(k,:)) - distance(s, m(j,:));
        Dis(h) = diffD;
        h = h+1;
%          if h>nPairs+1
%              error(); % needed only at debugging stage.
%          end
    end
end

% The final equation
for k=1:nPairs
    res(k)= Dis(k)./c - delays(k);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function answer = loc1ConditionVis(z)
% Helps visualization of the condition number of localize1 routine
% as a function of source coordinates.
% Due to our limitations to 3D visualization, user picks
% a plane where the sources will be examined upon, by specifying
% a z value.

% z -> The above-mentioned z value.

figure(3);
cond = [];
xValues = [];
yValues = [];
for xVal = -2:2;
```

```
    for yVal = -2:2;
        [a, cond(end+1)] = localize1(1, [xVal, yVal, z]);
        xValues(end+1)=xVal;
        yValues(end+1)=yVal;
    end
end

plot3(xValues, yValues, cond, '*');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

4.2 Customized Source Separation Code

DISCLAIMER:- The code that is written by James V. Stone, the author of Independent Component Analysis: a tutorial introduction is customized for our own use as follows. [5]

```
function answer = icaDemo()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
% Demonstration code for "Independent component analysis: A Tutorial
Introduction"
% JV Stone, MIT Press, September 2004.
% Copyright: 2005, JV Stone, Psychology Department, Sheffield
University, Sheffield, England.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
% CUSTOMIZED BY BINIYAM TESFAYE TADDESE.

% Basic Bell-Sejnowski ICA algorithm demonstrated on 2 speech signals.
% The default value of each parameter is given in [] brackets.

% [0] Set to 1 to hear signals.
listen=0; % set to 1 if have audio.

% [1] Set random number seed.
seed=9; rand('seed',seed);  randn('seed',seed);

% [2] M = number of source signals and signal mixtures.
M = 2;
% ### SET THESE OPTIONS! ###
simulation = 1;     % Are you working on real or simulated data?
gongChirp = 0;      % Set ONLY one of these to true. (Source types)
nonGausMonotones = 1;
simultaneous = 1;   % Are the signals mixed with a delay?
integerDelay = 0;   % default = 0;

% ######################################################################
% Load data, each of M=2 columns contains a different source signal.
% ######################################################################s
% Each column has N rows (signal values).
if simulation == 0
    % [1e4] N = number of data points per signal.
    N = 65000;
```

```
    % [10000] Fs Sample rate of speech.
    Fs=10000;
    % Load file
    x = load('KutirNoise.xlc');
 end

% ######################################################################
% Load standard matlab sounds (from MatLab's datafun directory)
% ######################################################################
% Set variance of each source to unity. WHY? (This contradicts
reality!)
% OBSERVATION: it doesn't quite work without normalizing the sources...
if simulation ==1
    if gongChirp == 1
        % $$$$$$$$$$ GONG + CHIRP
        % [1e4] N = number of data points per signal.
        N = 1e4;
        % [10000] Fs Sample rate of speech.
        Fs=10000;
        load chirp; s1=y(1:N); s1=s1/std(s1);
        load gong;  s2=y(1:N); s2=s2/std(s2);
    end
    if nonGausMonotones == 1
        % $$$$$$$$$$ NON GAUSSIAN MONOTONES
        % [65000] N = number of data points per signal.
        N = 65000;
        % [8000] Fs Sample rate of speech.
        Fs=1000;
        fMonotone1 = 300;  % Frequency = [1000 Hz];
        fMonotone2 = 150;  % Frequency = [100 Hz];
        domain1 = 0: (((fMonotone1).*2.*pi)./Fs) : ( floor(
(N./Fs).*(fMonotone1).*2.*pi ) );
        range1 = sin(domain1);  % frequency = 1000Hz

        domain2 = 0: (((fMonotone2).*2.*pi)./Fs) : ( floor(
(N./Fs).*(fMonotone1).*2.*pi ) );
        range2 = sin(domain2);  % frequency = 500Hz

        s(1, :) = zeros(1, N);
        s(2, :) = zeros(1, N);
        % Adjust the structure of the monotones as needed.
        for k = 1:N
            if mod(k, 5001) ==0
                s(1, (k-3000+1):k) = range1((k-3000+1):k)';
            end
            if mod(k, 10000) ==0
                s(2, (k-4000+1):k) = range2((k-4000+1):k)';
            end
        end

        % RESHAPE s!!! -> signals=column vectors.
        s = s'; s1=s(:,1); s2=s(:,2);
    end
```

```
    % Combine sources into vector variable s.
    s=[s1,s2];
    % Listen to speech signals ...
    if listen   soundsc(s(:,1),Fs); soundsc(s(:,2),Fs);end;

    % Make new MIXING MATRIX.
    % A=randn(M,M);
    A = [0.05, 0.09; 0.09, 0.05];
    % ########################3
    % 1. SIMULTANEOUS MIXING
    if simultaneous ==1
        % Make M mixures x from M source signals s.
        x = s*A;
    else
        % ########################
        % 2. NON-SIMULTANEOUS MIXING
        % Non instantenous sound propagation results in a delay
        % at each microphone that is imputed to the source
constelation.
        if integerDelay == 1
        % #########
        % Integer Delays.
            sMic1 = shiftSignals(s, [0, 0]);    % Pay attention to the
            sMic2 = shiftSignals(s, [0,0]);    % delay vectors!!
        else
        % #########
        % NON-Integer Delays.
            sMic1 = shiftSignalsNI(s, [0, 0.0645]);    % Pay attention
to the
            sMic2 = shiftSignalsNI(s, [0.06, 0]);    % delay vectors!!
        end
        % MIX
        xMic1 = sMic1*A(:,1);
        xMic2 = sMic2*A(:,2);
        % Form x
        x = [xMic1, xMic2];
    end
end

% Listen to signal mixtures signals ...
if listen   soundsc(x(:,1),Fs); soundsc(x(:,2),Fs); end;

% Initialise unmixing matrix W to identity matrix.
%W = eye(M,M);
W = randn(M, M);

% Initialise y, the estimated source signals.
y = x*W;

if simulation == 1
    % Print out initial correlations between
    % each estimated source y and every source signal s.
    r=corrcoef([y s]);
```

```
    fprintf('Initial correlations of source and extracted signals\n');
    rinitial=abs(r(M+1:2*M,1:M))
end

% ********************************************************************
maxiter=200;    % [100] Maximum number of iterations.
eta=.2;     % [0.25] Step size for gradient ascent.

% Make array hs to store values of function and gradient magnitude.
hs=zeros(maxiter,1);
gs=zeros(maxiter,1);

% Begin gradient ascent on h ...
for iter=1:maxiter
    % Get estimated source signals, y.
    y = x*W; % wt vec in col of W.
    % Get estimated maximum entropy signals Y=cdf(y).
    Y = tanh(y);
    % Find value of function h.
    % h = log(abs(det(W))) + sum( log(eps+1-Y(:).^2) )/N;
    detW = abs(det(W));
    h = ( (1/N)*sum(sum(Y)) + 0.5*log(detW) );
    % Find matrix of gradients @h/@W_ji ...
    g = inv(W') - (2/N)*x'*Y;
    % Update W to increase h ...
    W = W + eta*g;
    % Record h and magnitude of gradient ...
    hs(iter)=h; gs(iter)=norm(g(:));
end; W
% ********************************************************************

% PLOT change in h and gradient magnitude during optimisation.
figure(1);plot(hs);title('Function values - Entropy');
xlabel('Iteration');ylabel('h(Y)');
figure(2);plot(gs);title('Magnitude of Entropy Gradient');
xlabel('Iteration');ylabel('Gradient Magnitude');

if simulation ==1
    figure(3); title('Structure of -');
    subplot(3,M,1); plot(s1); title('source 1');
    subplot(3,M,2); plot(s2); title('source 2');
    subplot(3, M, 3); plot(x(:,1)); title('mixture 1');
    subplot(3, M, 4); plot(x(:,2)); title('mixture 2');
    subplot(3, M, 5); plot(y(:,1)); title('result 1');
    subplot(3, M, 6); plot(y(:,2)); title('result 2');
    % Scatter Plot
    figure(4); title('Scatter plot of -');
    subplot(3, 3, 1); plot(s1, s2); title('Source');
    subplot(3, 3, 5); plot(x(:, 1), x(:, 2)); title('Mixture');
    subplot(3, 3, 9); plot(y(:, 1), y(:, 2)); title('Result');
    % Histograms /~pdf
    figure(5); title('Histograms of -');
    subplot(3, M, 1); hist(s(:,1),50); title('Source1');
    subplot(3, M, 2); hist(s(:,2),50); title('Source2');
    subplot(3, M, 3); hist(x(:,1),50); title('Mixture1');
    subplot(3, M, 4); hist(x(:,2),50); title('Mixture2');
```

```
    subplot(3, M, 5); hist(y(:,1),50); title('Result1');
    subplot(3, M, 6); hist(y(:,2),50); title('Result2');
else
    figure(3); title('Structure of -');
    subplot(2, M, 1); plot(x(:,1)); title('mixture 1');
    subplot(2, M, 2); plot(x(:,2)); title('mixture 2');
    subplot(2, M, 3); plot(y(:,1)); title('result 1');
    subplot(2, M, 4); plot(y(:,2)); title('result 2');
    % Scatter Plot
    figure(4); title('Scatter plot of -');
    subplot(2, 2, 1); plot(x(:, 1), x(:, 2)); title('Mixture');
    subplot(2, 2, 4); plot(y(:, 1), y(:, 2)); title('Result');
    % Histograms /~pdf
    figure(5); title('Histograms of -');
    subplot(2, M, 1); hist(x(:,1),50); title('Mixture1');
    subplot(2, M, 2); hist(x(:,2),50); title('Mixture2');
    subplot(2, M, 3); hist(y(:,1),50); title('Result1');
    subplot(2, M, 4); hist(y(:,2),50); title('Result2');
end


% Print out final correlations ...
if simulation == 1
    r=corrcoef([y s]);
    fprintf('FInal correlations between source and extracted signals
...\n');
    rfinal=abs(r(M+1:2*M,1:M))
end

% Listen to extracted signals ...
if listen   soundsc(y(:,1),Fs); soundsc(y(:,2),Fs);end;
if simulation ==1
    answer = [s1, s2, x(:, 1), x(:, 2), y(:, 1), y(:, 2)];
else
    answer = [x(:, 1), x(:, 2), y(:, 1), y(:, 2)];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

4.3 Miscellaneous

4.3.1 Geometry of Aliases

This is the MATLAB program that I wrote in order to generate the hyperbolas depicting the geometry of the problem in *Chapter 2.1*.

```
function answer = DelayGeometry()
% AUTHOR BINIYAM TESFAYE TADDESE.
% It plots the locations of aliases when localizing
% sound sources using 2 microphones applying the
% cross correlation function to find the delay time.

% Microphone locations.
xm1=10; ym1=20;
xm2=5; ym2=10;

gridS = 40;

Lxs1 = [];  Lys1 = [];
LdelayS1 = [];

for xs1=0:gridS-1
    for ys1=0:gridS-1

        %xs2=10; ys2=10;

        dm1s1=sqrt( (xm1-xs1).^2 + (ym1-ys1).^2 );
        dm2s1=sqrt( (xm2-xs1).^2 + (ym2-ys1).^2 );

        delayS1 = dm1s1 - dm2s1;
        % Save the values for later plotting
        Lxs1(end+1)=xs1;
        Lys1(end+1)=ys1;
        LdelayS1(end+1)=delayS1;
    end
end
figure(1);
plot3(Lxs1, Lys1, LdelayS1); hold on;
plot3(xm1,ym1,0,'g*'); hold on;
plot3(xm2,ym2,0,'r*');

% Do a contour plot
figure(2);
cLdelayS1 = reshape(LdelayS1,gridS,gridS);
contour3(cLdelayS1); hold on;
plot(xm1,ym1,'g*'); hold on;
plot(xm2,ym2,'r*');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function answer = distance(a, b)
% implements the Euclidean distance formula to find
% the distance b/n the two points given.
x1 = a(1);
y1 = a(2);
z1 = a(3);
```

```
x2 = b(1);
y2 = b(2);
z2 = b(3);
answer = sqrt((x1-x2).^2+(y1-y2).^2+(z1-z2).^2);
```

4.3.2 Transforming the pdf of a signal

DISCLAIMER:- The entire code in *Chapter 4.3.2* is provided by Professor Daniel Kaplan.

```
function t = histxform(x,y,seed)
% histxform(x,y) transforms x so that it has the same histogram as y
% while maintaining the order of the ranks in x
% x and y should be single column vectors
% histxform(x,n,s) transforms x to have a histogram of the type specified
%     by n, with (optional) seed s.
%     n == 1 ==> N(0,1)
%     n == 2 ==> Uniform(0,1) --- this is deterministic
%     n == 3 ==> Triangular(0,2)

% Make sure there are some contents
if any([ length(x) == 0 , length(y) == 0 ])
    t = NaN;
    error('Empty time series given to histxform')
end

% see if we want a to specify a specific distribution
if any([length(y) == 1,nargin==3])
    if nargin==3
        randn('seed', seed);
        rand('seed', seed);
    else
        randn('seed', sum(100*clock));
        rand('seed', sum(100*clock));
    end
    if y == 1
        y = randn(size(x));
    elseif y == 2
        y = 0:1/length(x):1;
    elseif y == 3
        y = rand(size(x)) + rand(size(x));
    end
end

% convert everything to column vectors
[xr,xc] = size(x);
[yr,yc] = size(y);
```

```
if yr == 1
    y = y';
end
if xr == 1
    x = x';
end

% make sure that y is long enough
while length(x) > length(y)
    y = [y;y];
end

% Here's the meat of the algorithm
% sort x into ascending order
[z,zi] = sort(x);
% same with y
ys = sort(y(1:length(x)));
% ii will find the original indices in x of each point in sorted x
[z,ii] = sort(zi);
% x would be z(ii), but we want the values from ys so
t = ys(ii);

% if the original was a single row, return a single row
if( xr == 1 )
    t = t';
end
```

4.3.3 The Rejection / Acceptance Method

```
% AUTHOR: BINIYAM TESFAYE TADDESE.
% DISCLAIMER: Based on theory on [6] Chapter 7.

% This program produces a signal, which has a cdf characterized by the
% Hyperbolic Tangent Function, and hence with a pdf of square of the Hyperbolic
% Secant Function. The Bell Sejnowski ICA implementation assumes that
% a sound source signal is approximated by this distribution.

% N is the length of the uniformly distributed signal that we WANT to make.
% b == 1 -> Plot the results.


%
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@

% The Comparison Function used to simulate a Gamma Distribution could
% also be applied to simulate our distribution. The appropriate values
% of the parameters are listed below. The Comparison Function is:
% f(x) = c0 / { 1 + (x-x0)^2 / a0^2 }.
a0 = sqrt(2);
```

```
x0 = 0;
c0 = 1.5;
% The area under the Comparison Function is given by sqrt(a0)*c0.
% This area was made to be reasonably small.
area = c0.*sqrt(a0);

% Pick a uniformly distributed sequence from 0 to the area under the
% Comparison Function.
% The RATIO OF THE AREA under the sech(x).^2 TO our comparison function is
% 0.3. So, 70% of the random numbers we generate will be rejected! So, take
% this into consideration.
lengthOfSig = N; % copy N.
N = ceil((1./0.299).*(N));
U1 = rand(1, N);

% Pass the uniformly distributed numbers through the inverse of the
% Integral of the Comparision function. So, if we think of the Comparison
% Function as a pdf, we are passing the uniformly distributed numbers
% through the inverse of the cdf.
x = a0.*tan(pi.*U1);

% Pick another uniformly distributed numbers between 0 and f(x).
U2 = rand(1, N).*( c0 ./ ( 1 + ((x-x0).^2)./(a0.^2) ) );


%%% The CRUX of REJECTION/ACCEPTANCE METHOD.
% U1 and U2 form a pair of randomly generated numbers, which lie under
% the Comparison Function and are uniformly distributed.
% We need to REJECT those value of U2 that are above sech(x).^2 and ACCEPT
% those values of U2 that are below sech(x).^2. Because, sech(x).^2 is the
% pdf that is being used to simulate the signal out of.
% The corresponding members of x that are ACCEPTED will have a sech(x).^2
% distribution!!!

Ucomp = [U1; U2; x];

acceptIndex = find(Ucomp(2,:)<((sech(Ucomp(3,:))).^2));

U = x(acceptIndex);

% The exact length (0.30*N) of our signal needs to be known. So, trim it
% as necessary.
if (length(U) > lengthOfSig)
   U = U(1:lengthOfSig);
end

if b == 1
   % Plot the signal simulated out of the pdf, sech(x).^2.
   figure(1); plot(U); title('Signal with sech(x).^2 pdf');
   figure(2); hist(U, 1000); title('pdf of the signal, sech(x).^2');

   % Plot the Comparison Functino if needed.
   bbb = [-20:0.05:20];
   figure(3); plot(bbb, (c0 ./ ( 1 + ((bbb-x0).^2)./(a0.^2) )) ); hold on;
   plot(bbb, (sech(bbb)).^2 , 'r');
   title('The shape of the Comparison Function superimposed on sech(x).^2 (red)');
```

```
    % Return the signal with a sech(x).^2 distribution & its KURTOSIS, (if
    % needed). We also need to return a column vector.
    kurtosis = (mean((U-mean(U)).^4)./(std(U).^4)) -3;
end
res = U';
```

Chapter 5

Bibliography

[1] Blind Separation For Audio Signals Are We There Yet? (1999) Kari Torkkola

[2] Blind Separation Of Delayed Sources Based On Information Maximization (1996) Kari Torkkola

[3] Multichannel Blind Deconvolution: Fir Matrix Algebra And Separation Of Multipath Mixtures (1996) Russell H. Lambert

[4] A. Bell and T. Sejnowski. *An information-maximization approach to blind separation and blind deconvolution*. Neural Comp., 7(6):1129--1159, 1995.

[5] Independent component analysis : a tutorial introduction. Stone, James V.

[6] Numerical Recipes in C: The Art of Scientific Computing, William, Brian, Saul, William.

[7] Array signal processing : concepts and techniques. Johnson, Don H.