Blind Source Separation

Robert Gavelin
robert.gavelin.0531@student.uu.se

Harald Klomp
harald.klomp.3675@student.uu.se

Clinton Priddle
clinton.priddle.7851@student.uu.se

Mats Uddenfeldt
mats.uddenfeldt.9771@student.uu.se

June 11, 2004

Report for Adaptive Signal Processing Project
Signals and Systems group
Department of Engineering Sciences
Uppsala University
Sweden

Head of course: Mikael Sternad
Supervisor: Mathias Johansson

Abstract

Blind source separation (BSS) refers to the problem of recovering two or more sources from a number of unknown mixtures. This report presents a real-time implementation of blind source separation of two sources from two unknown mixtures. The algorithm used is a version of the degenerate unmixing estimations technique (DUET) algorithm and has been implemented in Matlab. We have tested the implementation on both artificial and real mixtures. The artificial mixtures could be separated almost perfectly, whereas only some of the "real recordings" produced a noticeable separation, others did not separate at all. We separated voice mixtures, but this algorithm works with any near W-disjoint orthogonal signals.

This report also contains a brief theoretical description of Bayesian ICA for static linear mixtures and the use of a feedback Infomax separation network for convolved mixtures.

All of the resources used in this project including Matlab code, wav-files, this report and more can be found on the project webpage at `https://student.signal.uu.se/~adapt0405/`.

Contents

1 Introduction

This report is written as part of the Adaptive Signal Processing Project given by the Signals and Systems group of the Department of Engineering Sciences at Uppsala University. Blind Source Separation (BSS) deals with the problem of separating unknown mixed signals without prior knowledge of the signals. There are many fields where Blind Source Separation could be useful, e.g. separation of radio signals in telecommunication, separation of brain waves from medical sensors or in audio applications as in hearing aids or for demixing stereo recordings. The methods for achieving BSS are not tied to any specific type of signals, but in this project mixed audio speech signals are used in the derivation and experiments.

The goal was to let two people talk simultaneously in two microphones placed some distance apart from each other. An adaptive algorithm would then separate the speakers, despite the fact that their speech is within the same frequency range and that no explicit desired signal is available to control the adaption. This problem is sometimes referred to as the "cocktail party"-problem. The work was originally intended to be a direct continuation of the corresponding project of 2003, which had investigated two signal separation problems: static linear mixing and convolved mixtures. A Bayesian approach to Independent Component Analysis (ICA) had been used to separate the static linear mixtures and a feedback Infomax separation network to separate convolved signals.

This year the goal was to find an algorithm that would perform the signal separation in real-time and implement it in Matlab. Early on in the project the Degenerate Unmixing Estimation Technique (DUET) was found while investigating new approaches to the problem. DUET was decided to be more interesting because it performs source separation by frequency domain processing and is independent of the number of mixed sources. We decided to put all our efforts into implementing DUET in Matlab and turn to the previously investigated methods only if we failed.

This report will begin with a brief theoretical introduction of the Bayesian ICA algorithm and the feedback Infomax separation network, but focus will lie on the DUET algorithm. The DUET algorithm was also successfully implemented and tested in Matlab.

2 Methods

2.1 Bayesian Independent Component Analysis

2.1.1 Bayesian source separation

Consider a number of sources $s_i(t)$, which are linearly mixed using a mixing matrix A with coefficients a_{ij} producing mixtures $x_i(t)$. The mixing equation can be written as

$$x = As \tag{1}$$

The aim of ICA is to find a separation matrix W that is the inverse of the mixing matrix A. The obvious limitation is that the number of sources must equal the number of mixtures to be able to calculate A^{-1}. The separated signals $u_i(t)$, are calculated as

$$u = Wx = WAs = A^{-1}As = s \tag{2}$$

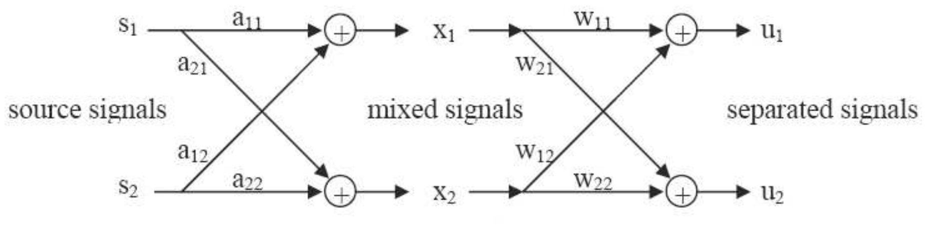


Figure 1: Mixing and separation network for two signals

The source separation function is derived using Bayesian probability theory. The probability of the model is described in terms of the likelihood of the data and the prior probabilities of the model and the data.

$$P(A,s|x,I) = P(A|s,I)\frac{P(x|A,s,I)}{P(x|I)} \tag{3}$$

where I represents any prior information and the probability of the mixing matrix is described in terms of the likelihood of the mixed signals. The full derivation of this algorithm can be studied in detail in [Knu99] and [ORS03]. Here follows a brief summary. By using the fact that the prior for x is a constant and the logical assumption that the sources are independent from the mixing matrix, (3) can be rewritten as

$$P(A,s|x,I) \propto P(A,s|I)P(x|A,s,I) = P(A|I)P(s|I)P(x|A,s,I) \tag{4}$$

The mixing matrix can be found by integrating over all possible values of the source signals and further simplified by taking the logarithm of the result

$$\log P(A|x,I) \propto \log P(A|I) + \log \int P(S|I)P(x|A,s,I)ds \tag{5}$$

A stochastic gradient method is used for finding the maximum of (5) with respect to the mixing matrix W

$$W_{i+1} = W_i + \Delta W \tag{6}$$

If A is orthogonal ΔW is found through

$$\Delta W = \frac{\partial}{\partial W} \lg P(A|x,I) = A^T + \frac{p_i'(u_i)}{p_i(u_i)}x^T = W + \frac{p_i'(u_i)}{p_i(u_i)}u^T W \tag{7}$$

To separate speech signals one need to find a suitable prior for the cumulative probability density functions. The hyperbolic tangent is generally considered suitable as prior for the amplitude distribution of speech. The hyperbolic tangent

$$g(u_i) = tanh(u_i) \tag{8}$$

gives

$$\frac{p_i'(u_i)}{p_i(u_i)} = -g(u_i) \tag{9}$$

which fits equation (7). Using the probability density function of the hyperbolic tangent together with the separation yields the following update algorithm for finding the separation matrix W:

$$\Delta W = W + \big(-g(u_i)\big)u^T W \tag{10}$$

$$W_{i+1} = W_i + \Delta W \tag{11}$$

2.1.2 Pre-processing

Pre-processing of the mixed signals can be used to enhance the performance of the Bayesian ICA algorithm as explained in [ORS03].

Centering a signal x_i is done by subtracting its mean vector $m = E\{x_i\}$ from the signal to make it a zero-mean variable. By centering the mixed signals x_i the algorithm is simplified, since it implies that u_i is calculated as a zero-mean estimate of the source signals. An estimate of the mean vector of the source signals can be added to u_i. The product of the separation matrix and the mean vector Wm gives this estimate.

Whitening is a way of decorrelating the source signals to create an orthogonal mixing matrix A. Instead of having to estimate an arbitrary n-by-n matrix, which would require estimating n^2 coefficients, an orthogonal matrix only has $n(n-1)/2$ degrees of freedom. By whitening A before the application of the ICA algorithm, but after centering, the observed vector x is transformed linearly to a whitened vector $\tilde{x}$. This whitened vector has uncorrelated elements and its covariance matrix equals the identity matrix

2.2 Separation network for convolved mixtures

2.2.1 Entropy Maximization

In a real case scenario it is highly unlikely to have a static linear mixture. Consider the case of a room with two people who are talking into two microphones. The mixing will not be static, since each voice will have a different delay to reach the microphones. In this case the mixing of two sources can be described as

$$x_1(n) = a_{11}s_1(n) + a_{12}s_2(n - D_{12}) \tag{12}$$
$$x_2(n) = a_{22}s_2(n) + a_{21}s_1(n - D_{21}) \tag{13}$$

Since the mixtures are convolved with each other and the delays are not known it is not as easy as in the static linear case to find a separation network. One approach that is effective is to minimize the mutual information between the components $f = g\big(u_i\big)$, where g is a function that approximates the cumulative density function of u_i. This is the equivalence of maximizing the entropy of f. In the case of speech the hyperbolic tangent can be used as a good approximation of the CDF, which, as shown in [ORS03], leads to the following update algorithm

$$W = W + \mu\big[-2tanh(u)\big]x^T \tag{14}$$

It is interesting to note that no deconvolution is performed to separate the convolved sources.

2.2.2 Infomax separation network

After finding an update algorithm which uses entropy maximization, we need to build a separation network to be able to implement the algorithm. Experiments have shown that the feedback network architecture shown in Fig. 2 has given the best results according to [Tor00]. The network has two FIR-filters, which are the cross-filters W_{12} and W_{21}. These filters add the convolution sums to the opposite branch in the network. This is done to avoid the whitening effects of entropy maximization, since the filters cannot remove time redundancies from their input signals. This way the redundancies between the signals are removed.

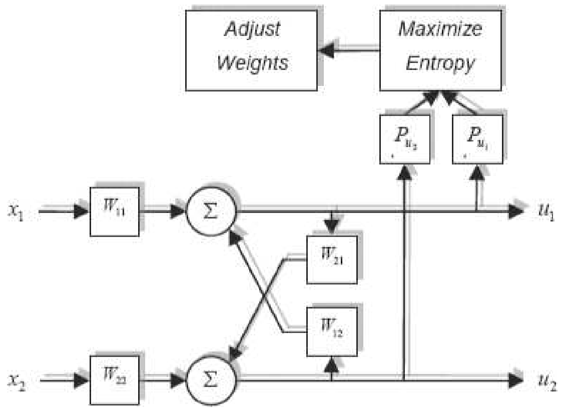


Figure 2: Infomax separation network

The mixtures are z-transformed to

$$X_1(z) = A_{11}S_1(z) + A_{12}S_2(z) \tag{15}$$

$$X_2(z) = A_{22}S_2(z) + A_{21}S_1(z) \tag{16}$$

which produces the following outputs from the separation network

$$U_1(z) = W_{11}X_1(z) + W_{12}U_2(z) \tag{17}$$
$$U_2(z) = W_{22}X_2(z) + W_{21}U_1(z) \tag{18}$$

By using 15 and 16 it is possible to derive the solution matrix W for perfect separation and deconvolution as

$$W = \begin{bmatrix} A_{11}(z)^{-1} & -A_{12}(z)A_{22}^{-1} \\ A_{22}(z)^{-1} & -A_{21}(z)A_{11}^{-1} \end{bmatrix} \tag{19}$$

Let us now return to the update algorithm 14 and adapt it to the separation network

$$W_{ij} = W_{ij} + \mu\big[-2tanh(u_i)\big]u_j \tag{20}$$

$$W_{ii} = W_{ii} + \mu\big[-2tanh(u_i)\big]x_i + \frac{1}{W_{ii}} \tag{21}$$

where ij denotes the cross-filter weights and ii the scaling factors. The learning algorithm have been split into the *decorrelation rule* of (20), which removes redundancies between the signals, and the *maximum entropy distribution rule* of (21), which tries to produce an output PDF as close to the flat unit distribution as possible. The extra term in (21) $1/W_{ii}$ is there to keep the algorithm from the situation where W_{ii} is so small that $-2tanh(U_i)x_i$ stays around the same value. The learning rules for the Infomax separation network are derived and further explained in [RJW02].

2.3 Degenerate Unmixing Estimation Techinque

2.3.1 Algorithm

To introduce the DUET algorithm we need to establish a new model for describing the mixing of sources. As before we will discuss the theory in regards to the specific example of two microphone channels. The sources, which in our case is represented by a number of persons talking in the room, are assumed to be standing on different locations, as shown in Fig. 3.

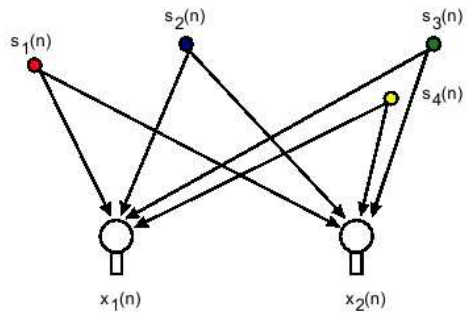


Figure 3: Two-channel microphone arrangement with multiple sources

The DUET algorithm operates in the frequency domain. No inverse matrix is calculated and it is one of the reasons it shows very good performance. Another bonus compared to Bayesian ICA is that the number of sources can be greater than the number of mixtures, in fact it can be used for an arbitrary number of sources. We worked with and modified the online DUET algorithm [RBR01], which has been developed especially with a real-time implementation in mind.

We chose to implement our own version of the online DUET algorithm because of the good results presented in [RBR01].

- It is fast. Their implementation is about 5 times faster than real time.
- It is effective. They achieved 15 dB average separation for anechoic mixtures and 5dB average separation for echoic mixtures.
- It separates an arbitrary number of sources from a set number of mixtures.

For a two-channel microphone arrangement with K sources, the incoming mixed signals x_1 and x_2 can be described as

$$x_1(n) = \sum_{j=1}^{K} s_j(n) \tag{22}$$

$$x_2(n) = \sum_{j=1}^{K} a_j s_j(n - \delta_j) \tag{23}$$

The mixtures x_1 and x_2 are sampled and split into blocks of length N with overlap. These sample blocks are multiplied with a windowing function W and then discrete time fourier transformed to

$$x^{\theta}_{1,2}(n) = W(n)x_{1,2}(n) \tag{24}$$

$$X_{1,2}(\omega) = \sum_{n=0}^{N-1} x^{\theta}_{1,2}(n)e^{-2\pi i n\omega/N} \tag{25}$$

Transforming the mixtures gives us a spectrogram with a two-dimensional time-frequency grid shown in Fig. 4. Since $x_{1,2}(n)$ consists of a mixture of the original sources $s_j(n)$, transforming the mixtures means that the sources now also have undergone a Short-Time Fast Fourier Transform (ST-FFT). We will refer to the fourier transform of the sources as $S_j(\omega)$. For a given source j we can describe the ST-FFT of (22) and (23) as

$$\begin{bmatrix} X_1(\omega) \\ X_2(\omega) \end{bmatrix} = \begin{bmatrix} 1 \\ a_j e^{-i\omega\delta_j} \end{bmatrix} S_j(\omega) \tag{26}$$

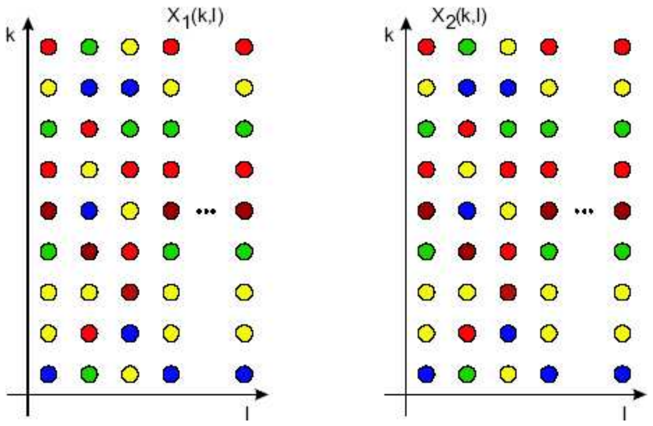


Figure 4: Time-frequency representations of the microphone signals

The DUET algorithm is based on the basic assumption that all of the sources have a sparse frequency spectrum for any given time. This implies that each time-frequency point in the spectrogram shown in Fig. 4 is associated with only one source. This property, which is essential for the DUET algorithm, is called the W-disjoint orthogonality property and can be described as

$$S_i(\omega)S_j(\omega) = 0, \qquad \forall i, j \qquad i \neq j \tag{27}$$

2.3.2 Parameters

To find the parameters in the online DUET algorithm, we will use an Maximum Likelihood (ML) gradient search. We begin by defining

$$\rho_j(\omega) = \frac{1}{1+a_j^2}|X_1(\omega)a_j e^{i\omega\delta_j} - X_2(\omega)|^2 \tag{28}$$

We can see that for any given source j there is a function ρ_j, which is 0 for all frequencies that belongs to j. That is

$$\rho_j(\omega) = 0 \qquad \forall\omega \in S_j(\omega) \tag{29}$$

As shown in [RBR01], the smooth ML objective function is

$$J = \min_{a_1,\delta_1,\ldots,a_K,\delta_K} \sum_\omega \frac{-1}{\lambda} ln\left(e^{-\lambda\rho_1} + e^{-\lambda\rho_2} + \ldots + e^{-\lambda\rho_K}\right) \tag{30}$$

where λ is the amplification factor.

The partial derivative of J with respect to δ_j is

$$\begin{aligned}\frac{\partial J}{\partial \delta_j} = \sum_\omega & \frac{e^{-\lambda\rho_j}}{\sum_{r=1}^K e^{-\lambda\rho_r}} \cdot \frac{2}{1+a_j^2} \cdot \omega a_j \cdot \\ & \cdot \Big(\mathrm{Im}\{X_1 e^{i\omega\delta_j}\} \cdot \mathrm{Re}\{X_2\} - \mathrm{Re}\{X_1 e^{i\omega\delta_j}\} \cdot \mathrm{Im}\{X_2\}\Big)\end{aligned} \tag{31}$$

and the partial derivative of J with respect to a_j is

$$\begin{aligned}\frac{\partial J}{\partial a_j} = \sum_\omega & \frac{e^{-\lambda\rho_j}}{\sum_{r=1}^K e^{-\lambda\rho_r}} \cdot \frac{2}{1+a_j^2} \cdot \\ & \cdot \Big(a_j|X_1|^2 - a_j\rho_j - \mathrm{Re}\{X_1 e^{i\omega\delta_j}\}\mathrm{Re}\{X_2\} - \mathrm{Im}\{X_1 e^{i\omega\delta_j}\}\mathrm{Im}\{X_2\}\Big)\end{aligned} \tag{32}$$

These partials were recalculated by us, since we failed to get the algorithm to function with the ones given in [RBR01].

We assume we know the number of sources in the mixtures and initialize an amplitude a_j and a delay δ_j estimate for each source. The parameters a_j and δ_j are updated based on the previous estimate and the current gradient as

$$a_j[k] = a_j[k-1] - \beta\alpha_j[k]\frac{\partial J}{\partial a_j} \tag{33}$$

$$\delta_j[k] = \delta_j[k-1] - \beta\alpha_j[k]\frac{\partial J}{\partial \delta_j} \tag{34}$$

where β is the learning factor and $\alpha_j[k]$ is a time and mixing parameter dependent learning rate for time index k and estimate j. It is practical to adjust the learning rate depending on the amount of mixture energy in the current estimate. The mixing energy can be described as

$$q_j[k] = \sum_\omega \frac{e^{-\lambda\rho_j}}{\sum_{r=1}^K e^{-\lambda\rho_r}} \cdot |X_1| \cdot |X_2| \tag{35}$$

and we define

$$qs[k] = \gamma qs[k-1] + q[k] \tag{36}$$

where γ is the forgetting factor. This allows us to write the parameter dependent update rate $\alpha j[k]$ as

$$\alpha_j[k] = \frac{q[k]}{qs[k]} \tag{37}$$

2.3.3 Demixing

According to (29), we know that ρ_j is minimum for any given time-frequency point that belongs to s_j. If this is not the case, the time-frequency point belongs to another source. We can therefore construct a time frequency mask based on the ML parameter estimator.

$$\Omega_j(\omega) = \begin{cases} 1 & \rho_j(\omega) \leq \rho_m(\omega) \qquad \forall m \neq j \\ 0 & \text{otherwise} \end{cases} \tag{38}$$

Now we can extract the discrete time fourier transform estimate of the jth source from mixture $X_1(\omega)$

$$\hat{S}_j(\omega) = \Omega_j(\omega) X_1(\omega) \tag{39}$$

This is represented by the *Demixing* block in Fig. 5.

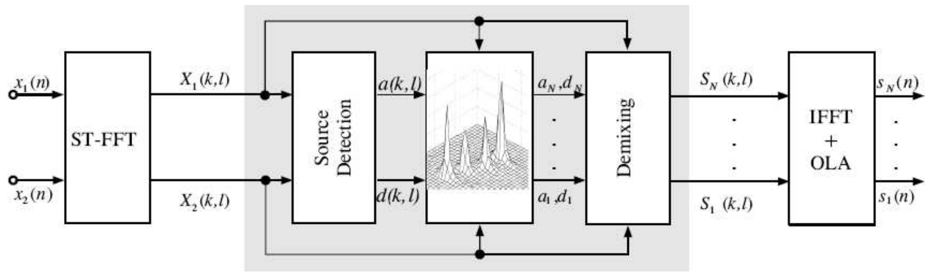


Figure 5: Block diagram of DUET source separation

At this point we have performed the signal separation and all that is left to do is to compute our windowed source estimate $\hat{s}_j^\theta$ using the inverse discrete time fourier transform

$$\hat{s}_j^\theta(n) = \frac{1}{N} \sum_{n=0}^{N-1} \hat{S}_j(\omega) e^{\frac{2\pi i}{N} n\omega} \tag{40}$$

3 Experiments

3.1 Equipment

Since the algorithm required input from two microphones simultaneously, these were connected to the line-in input of the soundcard via a microphone amplifier, so that one microphone connected to the left channel and the other one to the right channel. Matlab was used to capture the audio signals from the line-in input, separate the sources and output the separated data to the soundcard, according to the block diagram in Fig. 6.

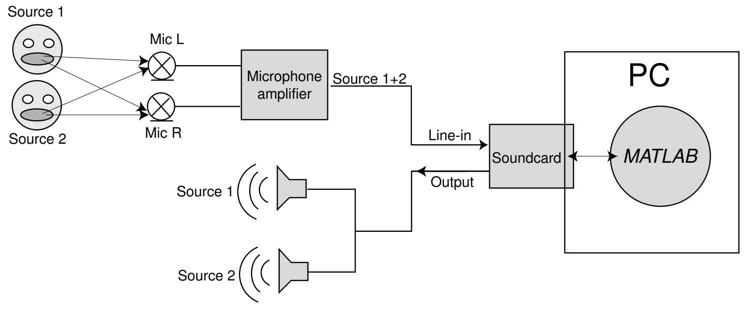


Figure 6: Block diagram of the equipment set-up

3.2 Matlab

3.2.1 Implementation of modified DUET algorithm

The algorithm is written in Matlab 6.5 and was run on a PC. See Fig. 7 for a program flow diagram. It works by taking a two-channel wav-file, with one mixture per channel, and reading it, or part of it, into an array. The program could easily be modified to take samples directly from the soundcard. It then takes 1024 samples at a time from this array and the fast fourier transform (FFT) for this set of samples is calculated. Once this is done we can estimate the parameter changes and update the parameters using equations (33) and (34). After updating the parameters, we can separate the signals from this block using a binomial mask as in equation (39). The separated two-channel array is then either saved to a wav-file or played by the soundcard. We then wait, if needed, until the buffer has enough elements, and start over with the next 1024 samples. This process is repeated until the whole file has been demixed.

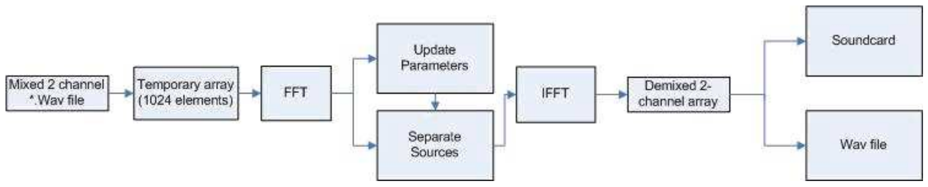


Figure 7: Program flow diagram

3.2.2 Data acquisition and playback

One big issue was how to acquire and play sound in real-time. The basic functions *wavrecord* and *wavplay* included in Matlab are good when dealing with large chunks of audio data. But since the BSS algorithm works continuously with small data blocks, there were many problems arising. It was for example impossible to perform other calculations while recording, since it is not possible to run *wavrecord* in a non-blocking mode. Our solution to this was to record everything in advance, save the data in a matrix using *wavread* and thereafter separate the matrix data in real-time. Fortunately, *wavplay* could be used to play the data in a non-blocking asynchronous mode, by using the flag *asynch*. Another problem was that the soundcard got overloaded, because the program tried to write to the soundcard too often. To solve this, we established a buffer for temporary storage of the output data, thus write longer data sequences to the soundcard less frequently. This made our code work on most computers, but it is still not an ideal solution, since soundcards with poor efficiency would still crash.

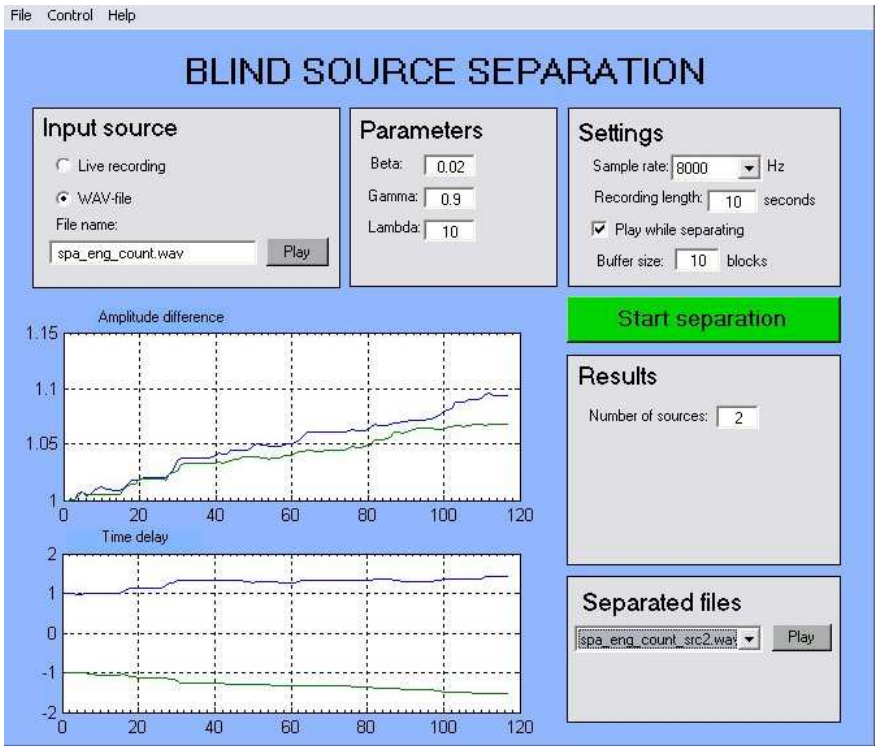
We also did a lot of research in the Data Acquisition Toolbox, which contains many functions to manage real-time acquisition and playback of audio data. Here the idea is to create one object for analog input and one for analog output. After setting the correct parameters, the objects can be started. The analog input object acquires data asynchronously (in contrast to wavrecord) and can be set to trigger a function after a certain number of samples. Our idea was to trigger a function every 1024 samples, perform the separation and write the result to the analog output object, which has an automatic buffer to solve the soundcard overload problem. Unfortunately we found out that the Data Acquisition Toolbox worked differently with different soundcards. Each and every soundcard has its own set of valid sample frequencies and a certain driver. For most Windows-based soundcard drivers, the adaptor *winsound* is associated with the soundcard. But the soundcard on the experiment audio workstation at the university did not support this, so we decided to drop this and just use *wavrecord* and *wavplay* instead.

Future development of this project requires a more stable solution to this problem, either by creating a better buffering solution for *wavplay* or finding a way to run Data Acquisition Toolbox in a general environment. Another method we discussed was to let some external program take care of the data acquisition and playback. There is for example Java support in Matlab, so that Matlab programs can collaborate with Java applications. Maybe a Java application would be able to handle the sound streams better. But we will leave this for following groups in this project course.

3.2.3 Graphical Interface

To facilitate sound recording, analysis of results and saving of separated data, we decided to develop a Graphical User Interface (GUI) for the Blind Source Separation application. This was done by using the user interface editor guide in Matlab. A screenshot of the GUI is shown in Fig. 8.

In the *Input source* box, the user can choose whether to use audio data from a sound file or to perform a live recording from the soundcard. It is possible to adjust the beta, gamma and lambda parameters to optimise the



BLIND SOURCE SEPARATION

Input source

Live recording

[✓ WAV-file]

File name: spa_eng_count.wav

Parameters

Beta: 0.02

Gamma: 0.9

Lambda: 10

Settings

Sample rate: 8000 Hz

Recording length: 10 seconds

[✓ Play while separating]

Buffer size: 10 blocks

Results

Number of sources: 2

Separated files

spa_eng_count_src2.wav

Figure 8: Screenshot of the user interface

performance of the algorithm. The user can also change other settings concerned with recording and playback of audio data. After pressing the *Start separation* button, the input data is acquired either from a file or the soundcard. Thereafter the algorithm separated the sources in real-time. After the separation is done, the amplitude difference and time delay between the sources are plotted in the window. One file for each source is also created and displayed in the *Separated files* box.

3.3 Parameters

We used the following parameters during our experiments; $\lambda = 10$, $\gamma = 0.9$ and $\beta = 0.02$ where λ is the amplification factor, γ is the forgetting factor and β is the learning factor. These parameters are similar to the ones used by [RBR01] in both an anechoic room and in an echoic office environment. As a windowing function we used a rectangular window. We found that different window-functions did not produce noticeably different results, but that the FFT size was important. The parameters are fairly stable if you choose a good step length. We used one step length for the artificial mixtures produced in Matlab and one for the real recorded mixtures.

3.4 Demixing of Artificial Mixtures

To get a source mixtures of audio files we used the Matlab function *wavread* to read the contents of a file into an array. By mixing two arrays we were able to get a mixture, which we could try to separate block-wise in real-time. The resulting demixed signals were saved into new arrays which could be stored to file using *wavwrite*.

3.5 Demixing of Real Mixtures

We had access to real speech mixtures used in various international BSS projects and also created a few of our own in the echo-free laboratory. Since the algorithm we implemented was supposed to run in real-time we read from the soundcard using the Matlab *wavrecord* function. We attempted to separate the resulting mixture block-wise in real-time. As with the mixed audio files the results were then stored to file.

4 Results

4.1 Demixing of Artificial Mixtures

The algorithm was able to demix all mixed audio files. Separation is very good and the algorithm converges within only a few iterations. Since we managed to demix all of the mixed audio files we will only present one case. The file *mymix.wav* consists of one person talking in Swedish and one person talking in Norweigian. The files were read separately and mixed within Matlab. The amplitude difference and time delay plots for *mymix.wav* are shown in Fig. 9. The resulting separated files can be found on the project webpage as *mymix_src*1*.wav* and *mymix_src*2*.wav*. As you can see we use a delay starting estimate of +1 and −1 which is the case of one source located to the right and one source located to the left. The algorithm converges very fast and the best separation contains almost no trace of the other source.

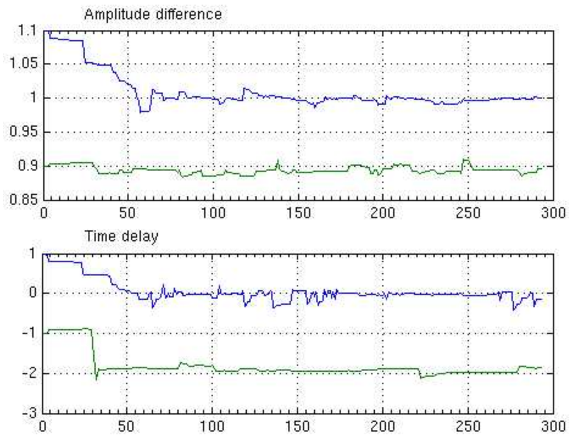


Figure 9: Amplitude difference and time delay plots for *mymix.wav*

4.2 Demixing of Real Mixtures

The real speech mixtures present a more difficult problem. Two speakers are recorded speaking simultaneously using a two-microphone setup. Since the algorithm was supposed to run in real-time, reading from the soundcard was done using the Matlab function *wavrecord*. We had some problems when writing to the soundcard in real-time because different soundcards only allowed a certain number of writes per second. For future implementations a better method must be found. An idea would be to use Java within Matlab or an external program for this.

In the first case of demixing real speech mixtures we used the file *spa_eng_count.wav*, which we found on [SR03]. The first speaker says the digits from one to ten in English (one, two, ...) and the second speaker counts at the same time in Spanish (uno, dos, ...). The recording was done in a normal office room and the distance between the speakers and the microphones is about 60cm in a square ordering. The resulting plots can be seen in Fig. 10. The time delay plot shows that the algorithm estimates the sources as further and further apart. At the same time the amplitude plot shows a strengthening curve for both signals. This file was separated successfully and the resulting files are available on the webpage as *spa_eng_count_src1.wav* and *spa_eng_count_src2.wav*.

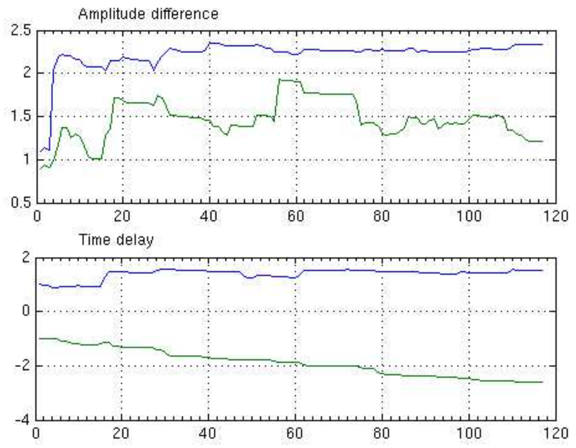


Figure 10: Amplitude difference and time delay plots for *spa_eng_count.wav*

Because of limited time we only made a few real recordings of our own. We had access to an echo-free environment at the university where we made a number of recordings of live speech using two speakers and a two-microphone setup. These recordings were much harder to separate. We chose to publish one of our recordings under the name *liverec.wav*. Even though we tried experimenting with different parameters and window-functions and sizes, we failed to get better results. The original and separated files are available on the webpage as *liverec_src1.wav* and *liverec_src2.wav*. As you can see in the plot in Fig. 11 the δ parameter never converges to something reasonable and stays around the starting value. Some more work needs to be done here to get better separation.

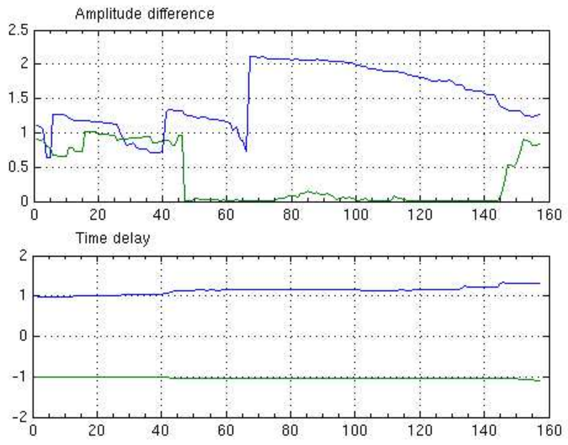


Figure 11: Amplitude difference and time delay plots for *liverec.wav*

5 Conclusions

We succeeded in implementing a version of the degenerate unmixing estimations technique (DUET) algorithm in Matlab. The implementation was tested real-time on both artificial and real mixtures. The artificial mixtures could be separated almost perfectly, whereas only some of the recordings of live speech produced a noticeable separation, others did not separate at all. As sources we used voice mixtures, but the DUET algorithm will work with any near W-disjoint orthogonal signals.

We spent a lot of time trying to get Matlab to handle the audio streams properly and did not have the time to come up with a solution. This is something that will need to be addressed in the future.

6 Further Studies

The work on implementing the DUET algorithm for Blind Source Separation should definitely continue considering the good results we managed to achieve. Here is a short list of items that we think could be subjected to further studies:

- Solving the audio stream problem by finding a way to run Data Acquisition Toolbox in a general environment or by using Java to handle the streams.
- Finding more stable parameters. They tend to jump around quite a lot if you use the wrong step length.
- Using some source location estimation technique to be able to find out why the algorithm fails to separate some mixtures. This could be done by calculating the angle and distance to the sources or by plotting histograms.

- Combining a frequency domain BSS algorithm, like DUET, with an algorithm operating in the time domain to see if the results can be improved.
- Experimenting with moving sources. One should not have to require the people talking to remain still.
- Experimenting with moving microphones. Someone requiring hearing aid equipped with BSS technology should be able to move around.

7 Resources

All of the resources used in this project including Matlab code, wav-files, this report and more can be found on the project webpage at `https://student.signal.uu.se/~adapt0405/`.

References

[Knu99] Kevin H. Knuth "A Bayesian Approach to Source Separation" in *Proceedings of the First International Workshop on Independent Component Analysis and Signal Separation: ICA'99* pp. 283-288

[Tor00] Kari Torkkola *Unsupervised Adaptive Filtering, Volume 1: Chapter 8, Blind separation of delayed and convolved sources* 2000: John Wiley & Sons Inc.

[RBR01] S. Rickard, R. Balan, J. Rosca *Real-Time Time-Frequency Based Blind Source Separation* December 2001: ICA2001 Conference, San Diego, CA.

[RJW02] R. Risberg, M. Janney, B. Warnquist *Adaptive Blind Source Separation* June 2002: Signals and Systems group, Uppsala University

[YR02] O. Yilmaz, S. Rickard *Blind Separation of Speech Mixtures via Time-Frequency Masking* November 2002: IEEE Transactions on Signal Processing

[ORS03] P. Ojutkangas, A. Runqvist, M. Sörnell *Blind Source Separation* June 2003: Signals and Systems group, Uppsala University

[BZ03] M. Baeck, U. Zölzer "Real-Time Implementation of a Source Separation Algorithm" from *Proc. of the 6th Int. Conference on Digital Audio Effects (DAFx-03), London, UK, September 8-11, 2003*

[SR03] S. Rickard http://princeton.edu/~srickard/bss.html *Blind Source Separation*