

DISTRIBUTED SOURCE CODING FOR AN ACOUSTIC SENSOR ARRAY

A Design Project Report

**Presented to the Engineering Division of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering (Electrical)**

By

Dennis Yueping Leung

Project Advisor: Professor Anna Scaglione

Degree Date: May 2006

Abstract

Master of Electrical Engineering Program
Cornell University
Design Project Report

Project Title: Distributed Source Coding for an Acoustic Sensor Array

Author: Dennis Yueping Leung

Abstract:

This project aims to demonstrate the effectiveness of distributed source coding on audio sources. Exploring a technique proposed by Azadeh Vosoughi and Anna Scaglione, an encoder and decoder for this application were built. This involves compressing an audio source by using cosets or bins and trying to recover it using available side information. Additionally, the use of multiple instances of signal source to suppress noise was investigated.

To accomplish the project goal, an acoustic sensor array was designed for the purpose of noise suppression. This sensor array was setup and used to collect experimental data to test the compression method. Then the data was encoded and decoded using a couple different structures. Performance calculations were done to demonstrate the usefulness of this compression method, even with real audio signals in practical applications, such as noise suppression.

Report Approved by
Project Advisor: _____

Date: _____

EXECUTIVE SUMMARY

The first step of this project was to learn about distributed source coding and the encoding and decoding method proposed by Azadeh Vosoughi and Professor Anna Scaglione in their paper “Precoding and Decoding Paradigms for Distributed Data Compression” [1]. To understand the process clearly, I implemented the scheme within Matlab. With this, I was able to run simulations and test the Matlab implementation. These simulations were run to see how the encoder and decoder work using various simulated audio sources. Simulated performance was also evaluated.

The next step was to test the method on real audio signals. In order to accomplish this, a physical experiment needed to be designed and setup. A number of designs and setups were proposed with a number of criteria in mind such as portability and synchronization. A final design using two way radios, microphones, and a data acquisition device was decided on and implemented. A four element sensor array was setup for the experiment. Additive noise, interference, and multipath cases were done in the experiment.

With the experimental data collected, the next step was to test the encoding and decoding. Preprocessing was needed including realigning data in time and normalizing. Finally this data was processed in Matlab to simulate the encoding and decoding. A couple methods using linear minimum mean squared error (LMMSE) weighted sums were implemented in the encoding and decoding to find an estimate of the source signal. The performance of these estimation methods were evaluated and compared.

TABLE OF CONTENTS

1 Introduction.....	5
1.1 Distributed Source Coding.....	5
1.2 DISCUS	5
2 Compression Structure.....	6
2.1 Encoder	7
2.1.1 Quantization.....	7
2.1.2 Cosets.....	7
2.2 Decoder	8
2.2.1 Equalization	8
2.2.2 Minimum Distance Detector.....	9
2.3 LMMSE Weighed Sum.....	9
2.4 Optimization of Delta	11
3 Acoustic Sensor Array Design.....	13
3.1 Criteria	13
3.2 Possible Solutions	14
3.2.1 Pocket PC Based Solution	15
3.2.2 802.11 Based Solution	16
3.2.3 Wire Based Solution	17
3.2.4 AM Transmission Based Solution	17
3.2.5 RF Remote Based Solution.....	18
3.3 Final Design: Two Way Radio Based Solution	19
4 Experiment.....	21
4.1 Physical Setup.....	21
4.2 Method 1	22
4.3 Method 2	23
5 Results.....	25
5.1 Time Delay and Distance Calculations.....	25
5.2 Performance	26
5.3 Additive Noise Case	30
5.4 Interference Case	31
5.5 Multipathing Case.....	32
7 Further Improvements and Applications	33
8 References.....	34
9 Appendix.....	35
9.1 Matlab Code.....	35

1 INTRODUCTION

1.1 DISTRIBUTED SOURCE CODING

Imagine a system that consists of number of acoustic sensors spread around a room recording its audio scene. Then these sensors transmit their information to a central processor to generate the best possible representation of the scene using all the received information collectively. Since these sensors are all in the same room, they are recording the same sound sources, thus are highly correlated. Due to this high correlation, much of the information sent from the sensors to the central processor is redundant. One could have each of the sensors communicate with the rest of the sensors to decide what information is pertinent to send, and what information is redundant to throw away. However, this intercommunication is very extensive in terms of computation and bandwidth, especially with a large sensor network.

According to Slepian and Wolf [2], if the system knows the joint distribution that determines the correlation model, sensors that do not communicate with each other should perform the same as a sensor network that does communicate with each other. Consider two correlated sources, X and Y . X is compressed at an encoder and recovered at a decoder with knowledge of side information Y . Slepian and Wolf state that this system can compress at the same limit as a system where Y is known at both the encoder and decoder. However, this theoretical lower bound has yet to be achieved practically. The subject of this kind of compression is called distributed source coding. Knowing their correlation structure, how much can one compress multiple independently encoded sources and still fully recover all the information at a decoder?

1.2 DISCUS

Pradhan and Ramchandran [3] try to address Slepian and Wolf's theoretical bounds by developing a practical method called DISCUS (Distributed Source Coding Using Syndromes). In their method, instead of sending the codeword representing X , the

codeword coset is sent and the receiver decodes X by choosing the codeword in the given coset that is closest to Y . Their method models the side information $y = x + n$.

Vosoughi and Scaglione [1] use DISCUS as a basis for their own distributed precoding and decoding scheme. Based on a side information linear model $y = Hx + n$, this method includes additional components such as linear transforms, equalization, and estimation. In this project, a simplified structure of their scheme is applied. Their method is especially formulated for vector sources. In the case of this project, the sources can be represented as scalar sources as audio signals are only one dimensional.

2 COMPRESSION STRUCTURE

In formulating this method, the following model of the received signal is used:

$$y = h * x + n$$

where h is the channel and y is the side information used. The block diagram below shows the components of both the encoder and decoder that will be implemented.

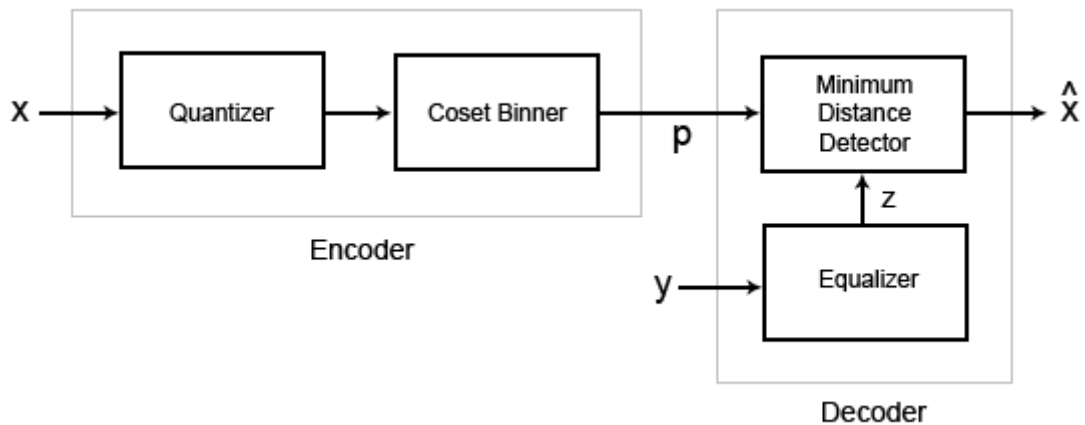


Figure 2.1: Block Diagram of Encoder and Decoder

2.1 ENCODER

2.1.1 QUANTIZATION

Due to the fact that a continuous range of data has to be sent at a finite rate, the encoder must quantize the source. This consists of dividing the data up into uniform intervals partitioned on the real line. Each sample of the source is represented by the centroid of the quantization interval the sample's value falls in. This is essentially choosing the minimum distance between the sample and all of the centroids. If one were to transmit the data immediately after quantization, each sample would be represented by Q bits, where 2^Q is the number of quantization intervals.

If one wanted to just send 1 bit per sample, the real line would be divided up into 2 intervals: all the negative values and all the positive values. This is an example of extreme compression, where almost all precision is lost. If one wanted to have minimal quantization error, a high number of quantization intervals can be used. However, a large number of bits must be transmitted to represent each interval. These are tradeoffs when dealing with quantization. Fortunately, in this application, the use of coset binning can be used as explained by Vosoughi and Scaglione.

2.1.2 COSETS

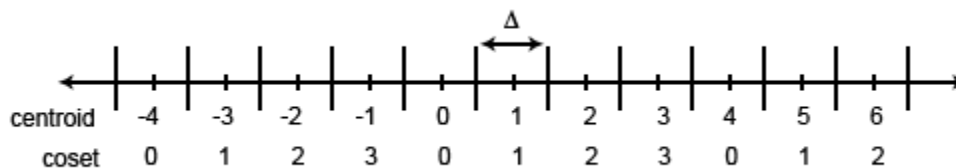


Figure 2.2: The real line divided into intervals of delta, with coset rate as 3

The next step is to take the quantization intervals and group them into cosets or bins. This means that the real line is first partitioned into quantization intervals of fixed length Δ and then partitioned again into a fixed number of cosets which are

interleaved among the quantization intervals. The number of cosets is defined by the coset rate. With increased coset rate, one would expect better performance at the decoder. However, a higher coset rate means more bits need to be sent, thus taking up more bandwidth.

While practical quantizers usual assign data over a finite range of values, the quantizer in this algorithm uses an infinite range, so there are no upper or lower bounds when quantizing the source. While this means the real line is partitioned into an infinite number of quantization intervals, the finite values of the source can still be mapped to cosets that are represented by a finite number of bits. To do this, each interval is defined by a set delta Δ . The cosets can be equated as the following:

$$c[n] = \text{round}\left(\frac{x[n]}{\Delta}\right)$$

$$p[n] = c[n] \bmod P$$

where c is the centroid index, P is the coset rate, and p is the coset for a given sample x .

Figure 2.2 shows an example of the real line being quantized into equal intervals of Δ , with all the corresponding centroid. A coset rate of 3 is used to partition each of these intervals. These assigned cosets are used to be transmitted to the decoder.

2.2 DECODER

2.2.1 EQUALIZATION

By modeling the side information as $y = hx + n$, there is a channel, h , between signals x and y . By incorporating an equalize, the side information can be reduced to $z = x + n'$, where n' is a new noise term. The equalizer essentially tries to find a channel filter h' to minimize $y - hx$. Once it does that, it applies the inverse of the found filter h' to try to best cancel out the effects of h . There are a number of methods of doing this such as the zero forcing or the least mean square methods. [4]

2.2.2 MINIMUM DISTANCE DETECTOR

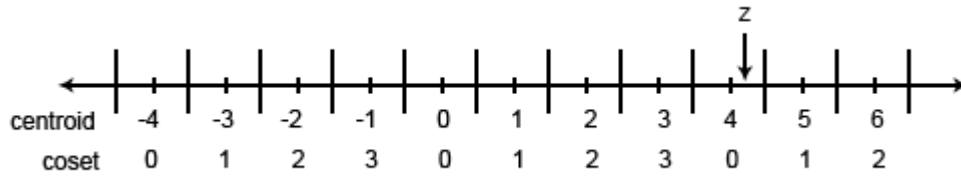


Figure 2.3: Minimum distance decoding

The decoder is where the coset values, corresponding to the encoded source, are received. Here, the cosets and the side information are used to attempt to recover the quantized source. The decoder looks at the side information and the corresponding received coset value. It then finds the quantized value that minimizes the distance between the side information value and all the quantization intervals corresponding to the received coset value. The decoding can be equated as followed:

$$\hat{q}[n] = \frac{z[n] - p[n] \cdot \Delta}{P \cdot \Delta}$$

$$\hat{c}[n] = \hat{q}[n] \cdot P + p[n]$$

$$\hat{x}[n] = \hat{c} \cdot \Delta$$

Figure 2.3 above can be used as an example. If the received coset is 1 and the side information is z as shown, then the decoded centroid would be 5.

2.3 LMMSE WEIGHED SUM

Given the described setup for a single encoder and its complementing decoder, one can extend the setup to include multiple encoders for different sources. Therefore, a sensor network can be setup with multiple remote and independent sensors along with the single central decoder with its side information. Using an array of sensors is deemed to be practical in many ways. Some applications include direction of arrival to sonar imaging to noise suppression. The main application used in this project is for noise suppression.

While not much can be done with a single received signal with noise, multiple recordings of a specific signal can be used to our advantage. Given that each of the added noise is independent of each other, summing and averaging all the received signals will essentially start to cancel the noise term. Since the signal of interest is coherent, it will be additive when summed from different sensors. However, the independent noise from each sensor aren't correlated, so when summed with other noise terms, they won't be additive. So the final result from averaging all the received signals from multiple sensors is an increase in signal to noise ratio.

However, what if you had knowledge of the signal to noise ratio of each recording? It would make sense to weigh a recording with a high signal to noise ratio more than a recording with a lower signal to noise ratio instead of just averaging the recordings. We want to optimize this summation by choosing the ideal weights to recover the source signal with minimal error. Here is a derivation for two received signals:

$$\hat{s}[n] = C_1 x_1[n] + C_2 x_2[n]$$

$$\text{where } \begin{aligned} x_1 &= s[n] + w_1[n] \\ x_2 &= s[n] + w_2[n] \end{aligned}$$

We would like calculate the linear minimum mean squared error (LMMSE) between the estimated s and actual s

$$\min_{C_1, C_2} E\left\{\left(\hat{s}[n] - s[n]\right)^2\right\}.$$

Expanding the MSE:

$$\begin{aligned} & E\left\{\left(\hat{s}[n] - s[n]\right)^2\right\} \\ &= \frac{1}{2} \left(C_2^2 \sigma_s^2 + C_2 C_1 \sigma_s^2 + C_2 \sigma_{w_2}^2 + C_2 C_1 \sigma_s^2 + C_1^2 \sigma_s^2 + C_1^2 \sigma_{w_1}^2 \right) - C_2 \sigma_s^2 - C_1 \sigma_s^2 + \sigma_s^2 \end{aligned}$$

Minimize by taking the partial derivatives and setting them to zero:

$$\frac{\partial MSE}{\partial C_2} = \frac{\sigma_s^2}{2} \left(C_2 + C_1 + \frac{C_1 \sigma_{w_2}^2}{\sigma_s^2} - 1 \right) = 0$$

$$\frac{\partial MSE}{\partial C_1} = \frac{\sigma_s^2}{2} \left(C_2 + C_1 + \frac{C_1 \sigma_{w_1}^2}{\sigma_s^2} - 1 \right) = 0$$

Solve for C_1 and C_2 :

$$C_1 = \frac{SNR_1}{SNR_1 + SNR_2 + 1}$$

$$C_2 = \frac{SNR_2}{SNR_1 + SNR_2 + 1}$$

$$\text{where } SNR_1 = \frac{\sigma_s^2}{\sigma_{w1}^2} \text{ and } SNR_2 = \frac{\sigma_s^2}{\sigma_{w2}^2}$$

The weights can be generalized to this:

$$C_i = \frac{SNR_i}{\left(\sum_n^N SNR_n \right) + 1}$$

2.4 OPTIMIZATION OF DELTA

There happens to be a way to determine the best delta to use, given a coset rate P.

The following equation calculates the mean squared error given a coset rate and delta:

$$E\left\{(\hat{x}[n] - x[n])^2\right\} = \left(\sum_n k[n]^2 \cdot \left[Q\left(\frac{(k[n] - 0.5) \cdot P \cdot \Delta}{\sigma_w}\right) - Q\left(\frac{(k[n] + 0.5) \cdot P \cdot \Delta}{\sigma_w}\right) \right] \right) P^2 \Delta^2 + \frac{\Delta^2}{12}$$

where $k[n] = \hat{q}[n] - q[n]$.

By numerical calculating the mean square error over a range of deltas, one can locate the optimal delta by finding when it's the minimum mean squared error. This delta results in the best performance when compared to other delta values. Therefore, the algorithm utilizes this numerical calculation to find the optimal delta to be used.

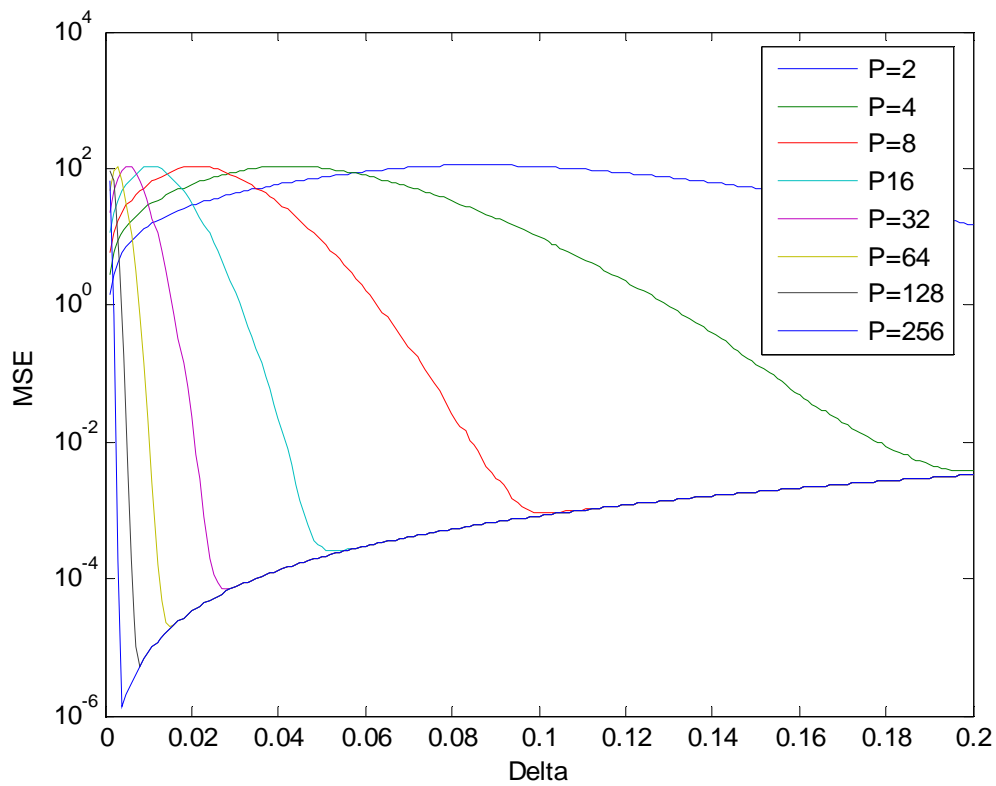


Figure 2.4: MSE versus deltas at $\sigma_w = 0.1$ at different coset rates R

Above is what the function looks like with different coset rates. As you can see, picking a delta too small will result in a large error, while picking a delta too big will result in a gradual increase of error. The minimum deltas are proportional to the variance of the noise.

3 ACOUSTIC SENSOR ARRAY DESIGN

Simulations were done in Matlab to test out how the compression works and corresponding performance. However, these simulations were done using ideal signals with perfect Gaussian noise. We are interested in how the compression method works with real signals being used in real and practical applications. So to do this, it was decided that a sensor array should be designed and built for experiments.

3.1 CRITERIA

In designing this experimental setup, there were a number of important criteria. First the sensors needed to be portable. This is due to the fact that the array should be flexible in setup. It should be able to be arranged in various configurations for different sensor array applications. Also, being portable makes the array capable of adapting to different environment or settings.

Another criterion for the design is the ability to be a long range away from the central decoder. The sensor array should be able to cover a large range or area. If the object is to record the scene in a room, the sensors should be able to be placed in all corners of the room. Since there is only one decoder, it will be inherently a distance away from various sensors. There shouldn't be any constraints on any reasonable range.

Next, the design should be easily expandable in terms of the number of the number of sensors in the array. While we may not need a large number of sensors for this project, it is helpful to design a setup where one can add additional sensors if needed. Sensor arrays generally get better performance with the more sensors used.

Most importantly, there is a question of simultaneous data recording or synchronization. When creating an effective sensor array, it is important to know the timing of a received signal relative to the timing of the received signals in all the other sensors. This requirement is crucial for the success of the sensor array.

There are a number of ways to solve this issue. One way is to start recording on all the sensors all at the same time. This means that a device or signal has to trigger all the sensors to instantaneously start, or at least have synchronized timed triggers.

Another solution is to have a centralized point to record all the data from all the sensors. Therefore all the sensors can continuously send data to this common point and have each stream of data recorded at the same time relative to each other. This method requires data to be acquired from multiple channels simultaneously.

One more way is to timestamp the recorded samples from each sensor. Each sample or each chunk of samples would be assigned a time of recording. For this to work, the clock or time of each sensor would have to be very accurately synchronized within microseconds.

Finally, there is the issue of money. The design should be cost effective in purchasing and building. Not only are we interested in the total costs of building an initial sensor array, but also the cost of adding additional units.

3.2 POSSIBLE SOLUTIONS

With all these issues kept in mind, a number of schemes were proposed. Exploring the possibility of each method, pros and cons were analyzed to decide on the most effective and reasonable design.

There are a number of common components that are used throughout a number of these solutions. They generally have the same function in each design. There is often an amplifier after the microphone. Because most microphones are electret and generally don't have much power, then produce signals on the order of milliamps. Therefore an amplifier is needed to get a strong enough signal from the microphone. The MCU component is a microcontroller. This is often after the amplifier when the signal needs to immediately be sampled. The use of the microcontroller is essentially for A/D conversion. It is generally a cheap way to do this conversion. It is also convenient to implement some other simple programs in the microcontroller if needed. Another common component is the data acquisition (DAQ) card. This is a device that takes multiple analog inputs and samples them "simultaneously" and sends the acquired data to a computer. The server component is usually simply a computer.

3.2.1 POCKET PC BASED SOLUTION

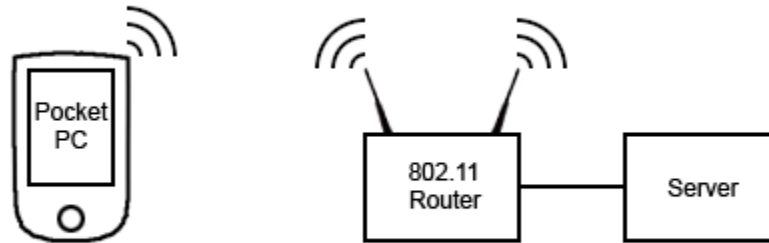


Figure 3.1: Pocket PC based solution

Because there were a couple Pocket PCs readily available for use, they were considered as an option. The Toshiba Pocket PCs were viable because they have 802.11 wireless Ethernet and a microphone integrated in it. The easiest implementation would have the server send out a trigger signal over 802.11 and have all the Pocket PCs start their record program simultaneously. Then the data would be stored on the Pocket PCs and downloaded to a computer manually to be processed. There are a number of hurdles with this design. First one has to learn the proprietary Pocket PC programming language to program it. This can be cumbersome as one has to program around Windows CE. Another issue is that the Pocket PC won't be dedicated to listening for the trigger and starting the recording. Just like any other computer with an OS, there are number of other processes and factors that may interfere. Therefore, timing may not be as precise as needed.

While there were some Pocket PCs available, which would mean there would essentially be no cost for the first couple units, the cost per unit after that would be very expensive. They run from around \$150-\$500 each.

3.2.2 802.11 BASED SOLUTION

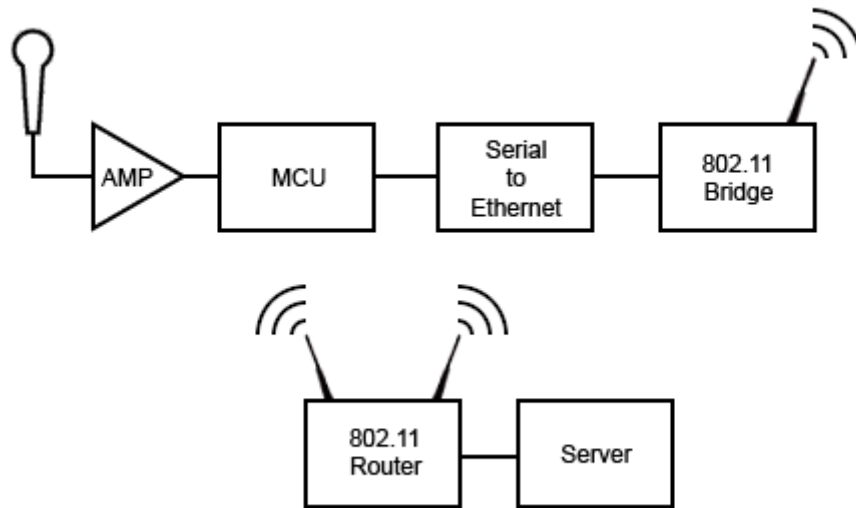


Figure 3.2: 802.11 based solution

This solution utilizes 802.11 wireless Ethernet as the medium of communication. Each sensor would require an amplifier, a microcontroller, a serial to Ethernet converter, and an 802.11 bridge. A serial to Ethernet converter is needed because the MCU only interfaces with serial inputs/outputs. With the 802.11 bridge, all the sensors can be part of a wireless network. This way, each sensor can stream data over the network at the same time while the server is simultaneously collecting the data and timestamping them. This implementation is pretty straight forward in terms of building. However, a bit of knowledge on TCP/IP will be required to organize the incoming packets.

The cost per unit can be quite pricey. The serial to Ethernet board would be around \$80. Then the cost of a wireless bridge would be around \$40. This is the major disadvantage with this setup.

3.2.3 WIRE BASED SOLUTION

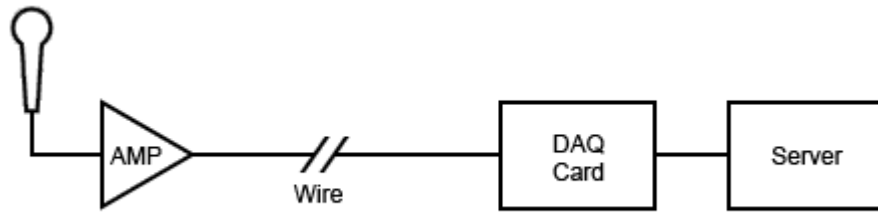


Figure 3.3: Wire based solution

This is a very simple and straightforward alternative design. Basically, instead of having wireless communication, all the sensors are connected by wires. These wires would be connected to a DAQ card and sampled then. However, this design neglects a couple important criteria. First being portability. If tethered by wires, the sensors aren't as flexible in placement. And most importantly, the wires limit the range of the sensors. The longer the wires, the harder they are to maintain.

The only major cost would be getting buying the DAQ card which would run around \$150. Then one can use multiple sensors per DAQ card.

3.2.4 AM TRANSMISSION BASED SOLUTION

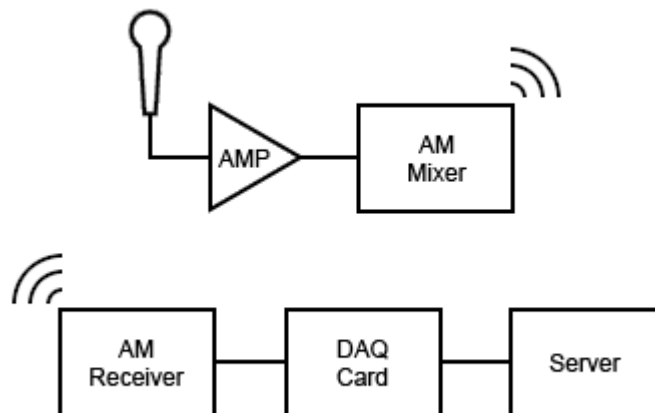


Figure 3.4: AM transmission based solution

This next design uses analog signals to transmit the data wireless. An AM transmitter and receiver were considered because it is fairly easy to implement in hardware. The data from the sensors would be transmitted using amplitude modulation. There would be a main receiver to convert the signal back to voltages. Then the signal would be sent to the DAQ card to be sampled and sent to a computer. There would have to be a pair of AM receiver and mixer per each sensor as they need to be different frequencies. The main problem with this implementation is the noise and reliability of the AM transmission. Another problem is the range may be limited especially with a less powerful AM transmitter.

Once again, the only major cost would be getting the DAQ Card. Then one can use multiple sensors per DAQ card.

3.2.5 RF REMOTE BASED SOLUTION

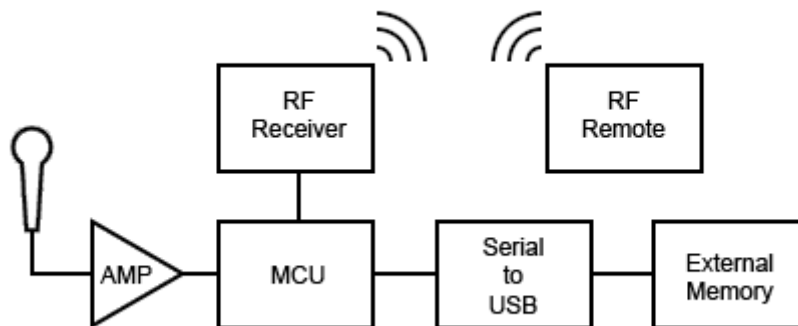


Figure 3.5: RF remote based solution

This solution utilizes RF. However, it doesn't transmit data over RF. It simply uses it to synchronize the start of recording over all the sensors. Basically, each sensor has a microcontroller with a RF receiver connected to it. When the RF receiver receives a trigger signal, it will tell the MCU to start sampling. Therefore, there's a single remote control to start all the sensors. There needs to be a serial to USB converter from the MCU as it only takes serial. USB is a very useful interface as it's compatible with many

devices. It would be used to transfer data to some memory point. One idea is simply use a laptop. However, if the setup has many sensors, a laptop per sensor wouldn't be practical. So another memory device should be used. Some knowledge of USB interfacing would be needed.

The major cost here would be getting the RF devices which run around \$20 for a set. It would also have to be programmed which may require a development board. A serial to USB converter run at about \$20. Finally, there's the external memory which would first substituted by a laptop.

3.3 FINAL DESIGN: TWO WAY RADIO BASED SOLUTION

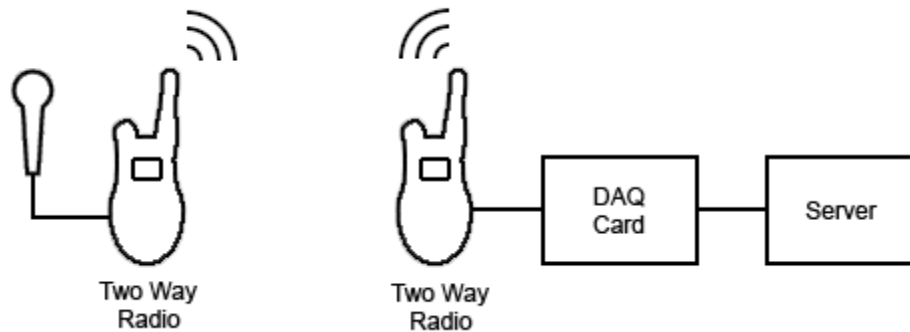


Figure 3.6: Two way radio based solution

This is another design that uses a wireless analog transmission. However, with two way radios, they use frequency modulation. Because they are two way radios, each device can receive and send data. So one would be dedicated for transmitting and another would be dedicated for receiving. Often two way radios have microphone input plugs and speaker output plugs as well. Using the output, the receiving radio would be connected to a DAQ card which would sample the signal into a computer. Also, each radio must be capable of being on their own frequency.

As for costs, initial setup costs would include the DAQ card which runs about \$150. However, once that's obtained, the price per unit should be much lower. After weighing all the options, it was decided to choose the two way radio based solution. It fits all the criteria quite nicely. They are very portable and easy to move around. They

have excellent range, some advertised to have 10 mile range (much further than needed). The synchronization issue is solved by using a DAQ connected to a computer. The setup is also easily expandable as one can just buy more sets of radios and simply hook them up to the DAQ card. Finally, they are relatively low cost.

The following are the major components used for the sensor array:

Midland LXT305 Two Way Radio

- 22 GMRS/FRS Channels
- Operating Frequencies: UHF 462.5500 ~ 467.7125 MHz
- Microphone Input
- Speaker Output
- \$15

Measurement Computing USB-1208FS Data Acquisition Device

- 4 differential/8 single-ended analog inputs
- 12-bit (Diff.)/11-bit (SE) resolution
- Up to 50 kilosample/second sample rate
- 2 12-bit analog outputs
- 16 bits of DIO
- USB Connection
- Matlab Compatible
- \$150

Logitech Desktop Microphone

- Sensitivity: -67dB/ubar, -47dBV/Pascal +/-4dB
- Frequency response: 100-16kHz
- \$20

Wires, batteries, and audio plug adaptors were also needed. Data generation and collecting were done on laptops using Matlab.

4 EXPERIMENT

4.1 PHYSICAL SETUP

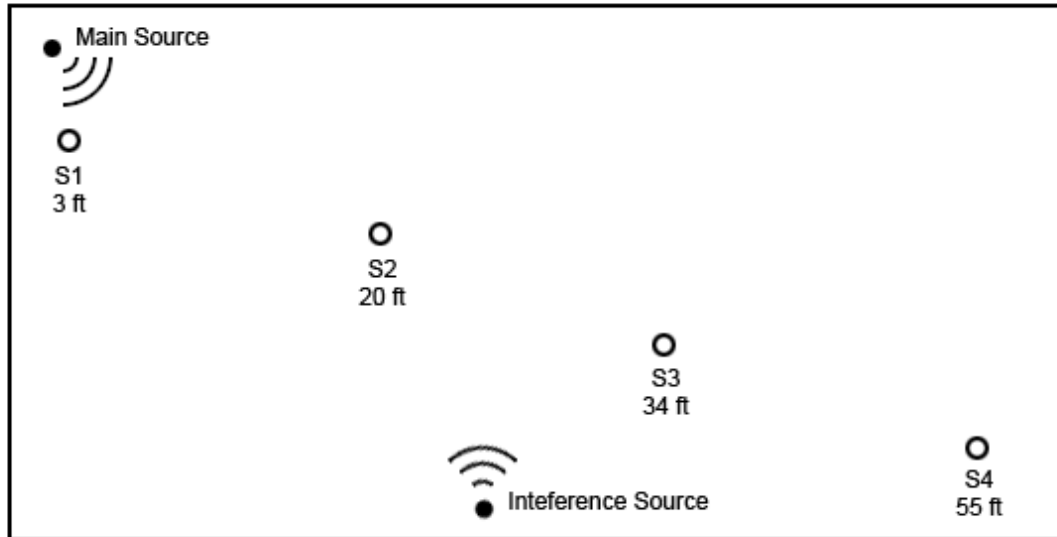


Figure 4.1: Experiment layout with measured distances from main source

For this experiment we purchased four sets of two way radios to construct a four element acoustic sensor array. They were placed to be spread out in a large room where S1 and S4 were as far apart as possible. This was done to get different types of attenuation from each sensor and analyze the effects on the performance of the algorithm. Sampling rate of each channel on the DAQ card was set at 10,000 Hz. This is sufficient for audio signals.

For the source signal, speakers hooked up to laptops were used. There were two different sources. One was the main source of interest. Another source was used for cases of interference and multipathing. This source was located at another position of the room. Speech was simulated as the signal from the source as it is a practical situation to have the human voice as the signal of interest. Speech was also used at the interference source. For simulating interference, different speech was used. However, to simulate multipathing, the same speech as the main source was used (with a time delay).

There are some things to note about the experimental setup. The audio signals sampled and recorded are all positive values. Therefore the raw samples have a positive DC offset. This data needs to be preprocessed to find the DC value and offset it to zero. Another thing to note is the radios also have a DC offset which actually changes over time. At the moment one holds down the button to transmit to the other radio, the DC value decays. Fortunately the decay is exponential. After a few minutes of leaving the radio on, the DC value ends up at a stable value.

4.2 METHOD 1

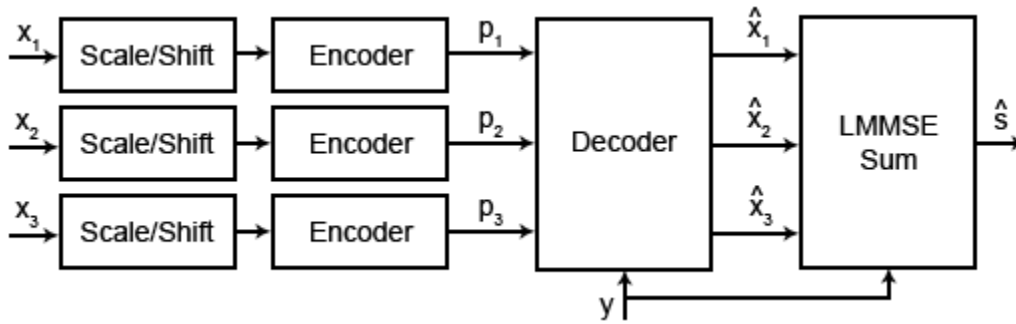


Figure 4.2: Method 1 block diagram

The first compression method can be seen in the above figure. Three of the sensors will have their data compressed while a fourth sensor will be used as side information. All the data must be time shifted before encoding and decoding so they line up with each other. The following model will be used:

$$x_i[n] = A_i s[n] + w_i[n] \quad \text{for } i = 1, 2, 3, 4$$

where s is the signal source, and w is zero mean additive white Gaussian noise with variance $\sigma_{w_i}^2$. The noise in these radios is very apparent as analog transmissions are susceptible to noise (the static). The variance of the noise $\sigma_{w_i}^2$ can be found by simple calculating the variance of the recording when there is no source signal.

To estimate the source better, the model needs to be reduced to

$$x'_i[n] = \frac{x_i[n]}{A_i} = s[n] + w'_i[n]$$

where $w'_i[n]$ now has a variance of $\frac{\sigma_{wi}^2}{A_i^2}$. So to find A_i , we can use the fact that the

noise is independent to the source signal:

$$\sigma_{xi}^2 = A_i^2 \sigma_s^2 + \sigma_{wi}^2$$

$$A_i^2 = \frac{\sigma_{xi}^2 - \sigma_{wi}^2}{\sigma_s^2}$$

This A_i will be used to scale each received signal accordingly.

After this scaling and time shifting, the three signals are compressed using the encoder and then uncompressed at the decoder. These three decoded signals are then combined using the weighted sum calculations to best estimate the source signal.

4.3 METHOD 2

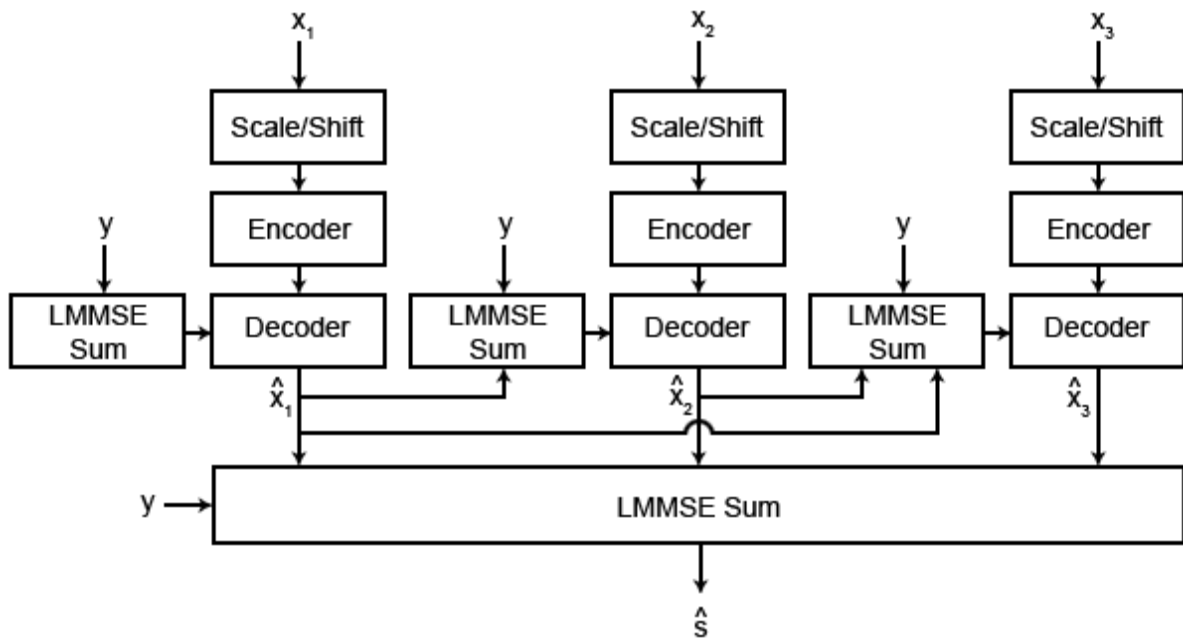


Figure 4.3: Method 2 block diagram

A second compression method for estimating the source involves compressing each received signal one at a time in an iterative process. Instead of using an individual side information source, it uses previously processed signals to improve the side

information. This improves the error between the side information and the signal that needs to be decoded. Given received signals (after normalization and time shifts as described above):

$$\begin{aligned} y &= s + w_y \\ x_1 &= s + w_1 \\ x_2 &= s + w_2 \\ x_3 &= s + w_3 \end{aligned} \quad \text{where } \sigma_s = 1$$

Form estimate x_1 from y by finding its LMMSE coefficient:

$$\min_C E\left\{\left(\tilde{x}_1 - x_1\right)^2\right\} \text{ where } \tilde{x}_1 = Cy \text{ and } x_1 = y - w_y + w_1$$

$$\text{which comes out to } C = \frac{1}{1 + \sigma_{wy}^2}.$$

To use the optimal delta, the variance of the new side information, $\tilde{x}_1 = x_1 + \varepsilon_1$, must be

$$\text{found: } \text{VAR}\{\varepsilon_1\} = E\left\{\left(\tilde{x}_1 - x_1\right)^2\right\} = 1 + \sigma_{w1}^2 - \frac{1}{1 + \sigma_{wy}^2}.$$

With all this information, x_2 is reconstructed to \hat{x}_2 . This ends the first iteration. The second iteration goes through the same process, except \hat{x}_2 is used for the side information.

Form estimate x_2 from both y and x_1 by finding the LMMSE coefficients of the sum:

$$\min_C E\left\{\left(\tilde{x}_2 - x_2\right)^2\right\} \text{ where } \tilde{x}_2 = C_y y + C_1 x_1$$

$$\text{This solves to } \begin{bmatrix} C_y \\ C_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 + \sigma_{wy}^2 & 1 \\ 1 & 1 + \sigma_{w1}^2 \end{bmatrix}^{-1} \text{ and comes out to}$$

$$C_y = \frac{\sigma_{w1}^2}{\sigma_{w1}^2 + \sigma_{wy}^2 + \sigma_{w1}^2 \sigma_{wy}^2} \text{ and } C_1 = \frac{\sigma_{wy}^2}{\sigma_{w1}^2 + \sigma_{wy}^2 + \sigma_{w1}^2 \sigma_{wy}^2}.$$

Now the variance of the new side information, $\tilde{x}_2 = x_2 + \varepsilon_2$, must be found:

$$\text{VAR}\{\varepsilon_2\} = E\left\{\left(\tilde{x}_2 - x_2\right)^2\right\} = 1 + \sigma_{w2}^2 + \text{sum}(C).$$

The process continues until all the signals are used. Once that finishes, an overall LMMSE weighted sum is calculated to get the final estimation of s . This can be done as shown in the LMMSE section.

5 RESULTS

5.1 TIME DELAY AND DISTANCE CALCULATIONS

For the purpose of our sensor array, the time delay of each received signal is very important. So to initially check the validity and quality of the data collected, one can measure the arrival timing at each sensor and calculate their relative distances. The cross correlation can be used to find timing differences between two received signals. It is basically a function of relative time between the signals:

$$r_{xy}[n] = x[n] * y[-n] = \sum_k x[k]y[k + n]$$

It can be thought of as a sliding dot product or time reversed convolution. At the time where the two signals are coherent, their cross correlation will result in a spike or jump in magnitude which corresponds to the maximum value of the function.

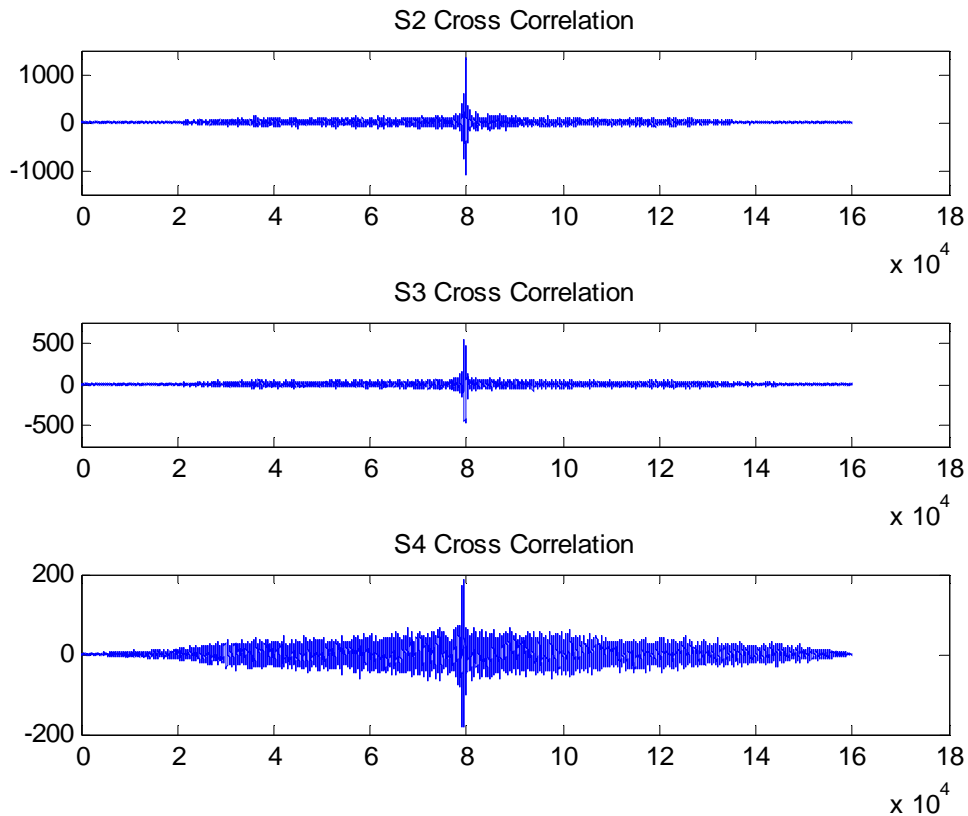


Figure 5.1: Cross Correlations with S1

So, if one signal is delayed relative to the other signal, there will be an equal delay of the maximum value in their cross correlation. Since the data collected are simply samples, the delay will be in terms of samples. So to get the actual time delay, simply divide the number of samples by the sampling frequency, 10,000 Hz. Knowing the speed of sound is approximately 340 meters per second, the final distance can be calculated. The distances calculated (and an average over three data sets) are as followed:

	S2	S3	S4
Data Set 1	20.8596	40.3806	60.3556
Data Set 2	20.8596	40.3806	57.5591
Data Set 3	20.8596	40.3806	57.6706
Average	20.8596	40.3806	58.5284

Figure 5.2: Calculated distances (feet) from S1 using cross correlations.

The measurements are extremely precise when it comes to S2 and S3. There are some variations with S4, but that is to be expected as it is further away from the others, thus lower signal to noise ratio. As you can see, they are fairly consistent to the distances manually measured as stated in the experimental setup section. However, they are not exact. This could be due to a number of things. First, the speed of sound, 340 meters per second, is an approximation and can vary depending on elevation/pressure and temperature. Another source of error is the accuracy of the manual measurements.

5.2 PERFORMANCE

In doing this experiment, it is important to see how well the compression method works. To make sure an individual pair of encoder/decoder is working effectively, error analysis was done. Taking a single sensor and it's received data x , it was encoded then decoded as \hat{x} . Another sensor's data was used as the side information y . This data was

encoded and decoded at different coset rates along with its corresponding mean squared error between \hat{x} and x .

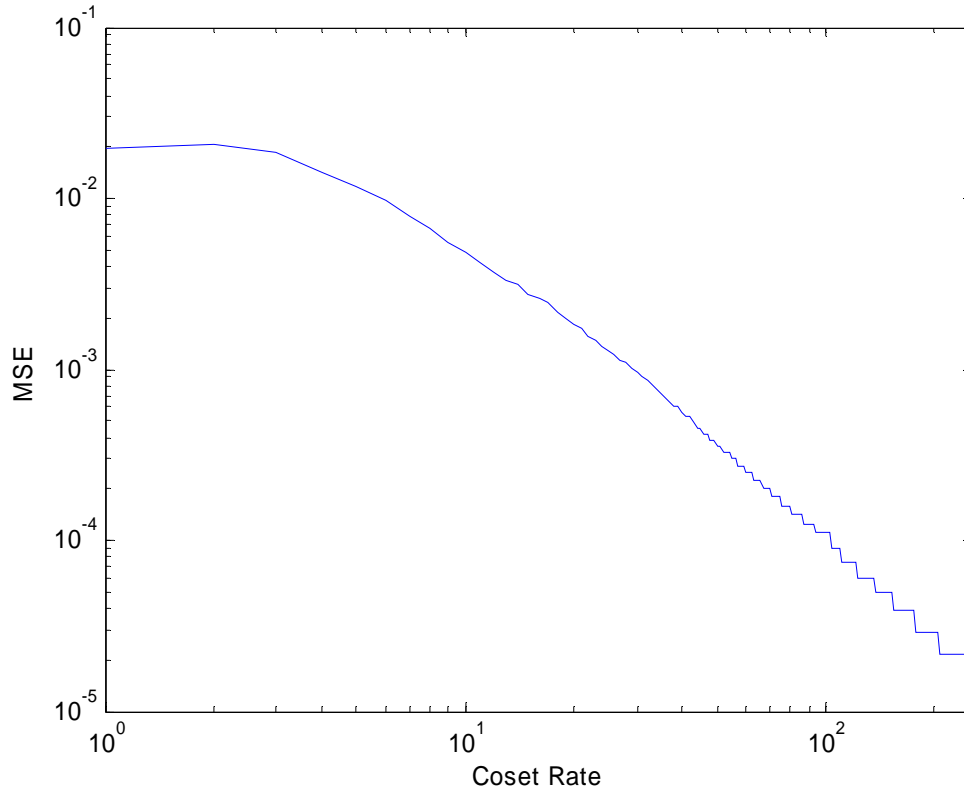


Figure 5.3: MSE of x estimation

This plot of the MSE shows what is expected. With increased coset rate, the MSE decreases. It can be seen as a linear decrease in logarithmic scale. The error should eventually converge to the quantization error which is a function of delta: $\frac{\Delta^2}{12}$. There is a bit of jittering as the graph goes towards higher coset rates. I suspect this is due to the sensitivity in choosing deltas at those points as the optimal deltas are considerably small.

Next, the performance of the whole algorithm is analyzed. These include Method 1 and Method 2 as described above. We are interested in how well these methods estimate the source of interest. We would also like to know which method performs better.

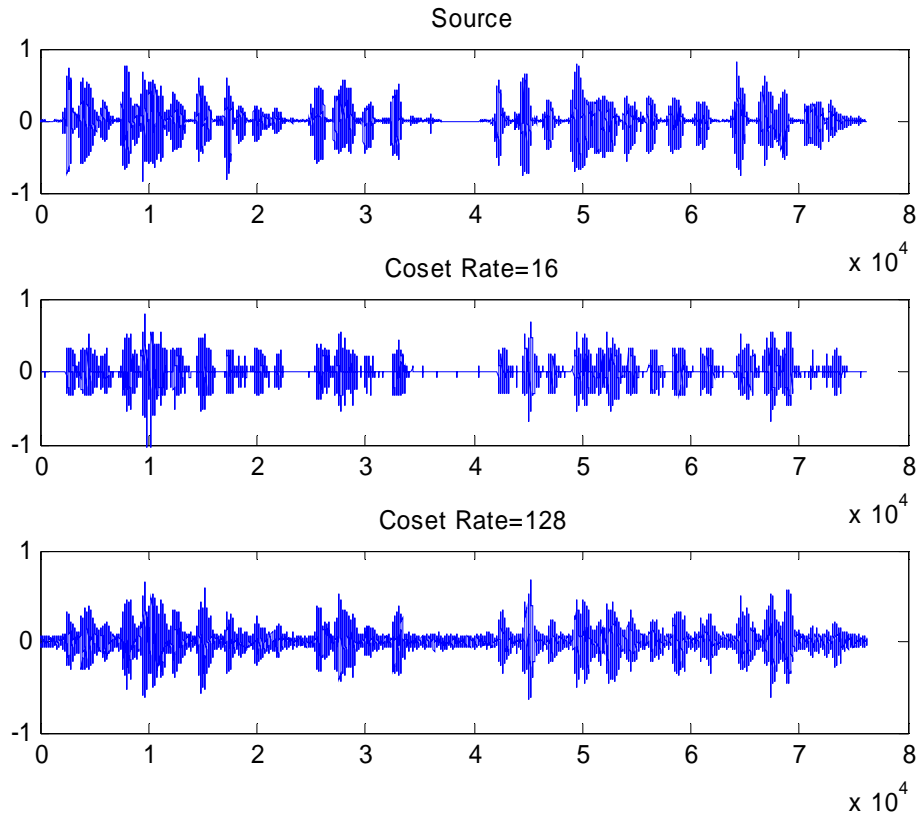


Figure 5.4: Example outputs at different coset rates

The source (a speech signal) can be seen at the top part of Figure 5.4. Examples of the output estimates s are shown. The one with a lower coset rate has noticeable errors with a number of spikes and jagged representations. The bottom signal uses a higher coset rate which resulted in a better representation of the signal. However, notice that it still includes some the captured noise. If actually listening to the signal of low coset rate, one would hear many artifacts, distortion, and noise. However, once a sufficient coset rate is used, the audio usually sounds normal (besides some added noise from the sensors).

The MSEs were calculated for both Method 1 and Method 2 as coset rates were increased. The performance can be seen in the plot below. While both methods effectively decrease in error with increased coset rate, Method 2 is noticeably more efficient. This means with one can get equivalent performance will a lesser number of bits using Method 2. This is actually expected because the side information was

optimally adapted to decrease the difference between the side information and decoding signal. This means that less coset bits are needed to recover the signal.

Since the DAQ device quantizes a continuous analog input at a precision of 11 bits, (2048 values), this is the maximum quantization rate the algorithm can use. So performance can't be greater than this rate. To find the upper bound of the performance (lower MSE bound) of the algorithms, the quantization and coset rate was set to 2048. The MSE found at this point was 0.02382 or $10^{-1.622}$ (in estimating s). This value was essentially the same for each algorithm. As you can see in the figure below, each of the methods are converging to this lower bound.

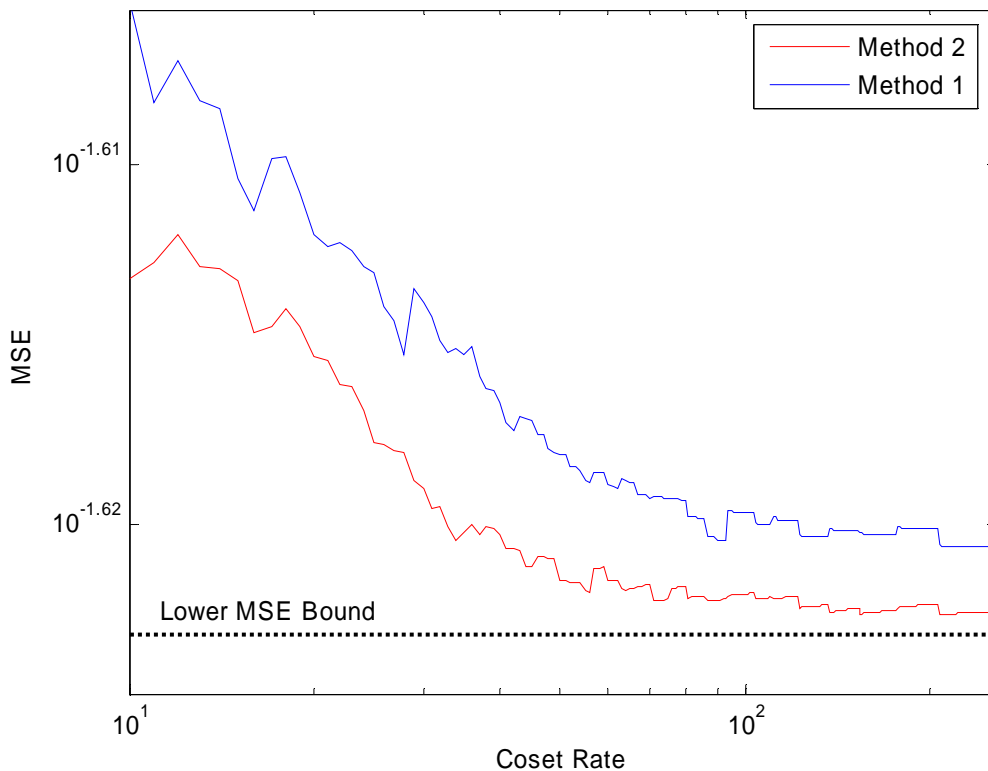


Figure 5.5: MSE of s estimation for both methods with lower bound

Finally, addressing the overall question of distributed source coding
 How does the compression performance of distributed source coding compare to simple quantization compression? To see, the data from each sensor was compressed by using normal scalar quantization instead of using any DSC algorithm. This data from each

sensor is combined once again using the LMMSE weighted sum to best estimate the source. The results are seen in Figure 5.6. The performance is much worse than either of the two DSC methods. It is clear that there is much to be benefited from using a DSC algorithm in these applications.

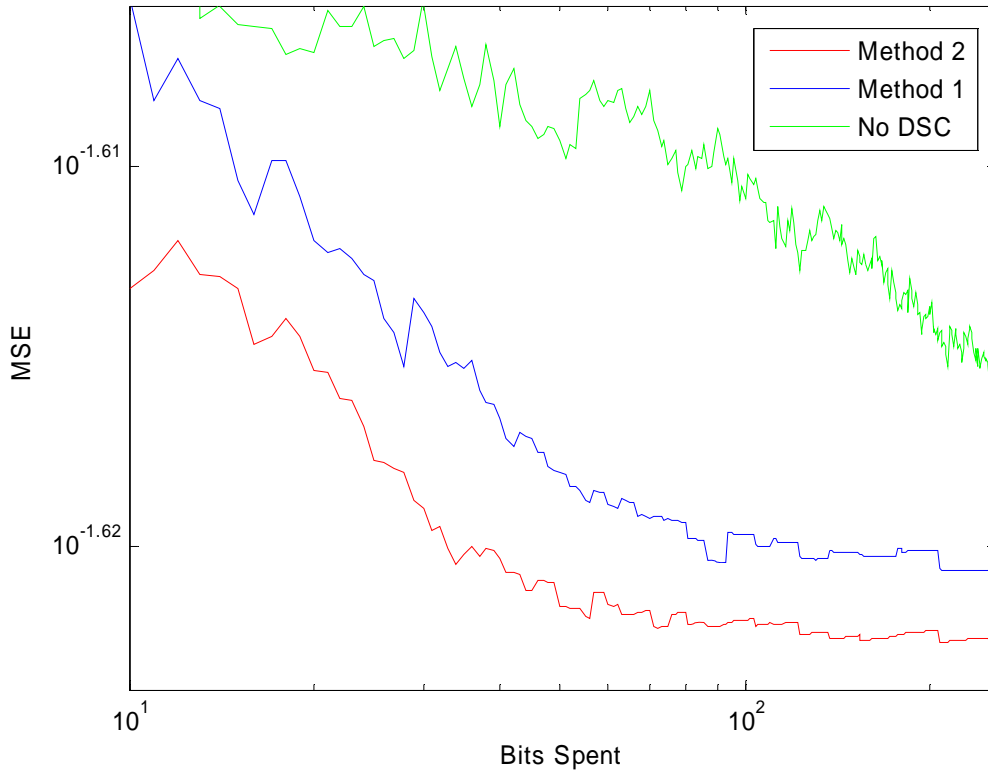


Figure 5.6: Performance comparison between using and not using DSC.

5.3 ADDITIVE NOISE CASE

When collecting data during the experiment, a case where only a single source was used. Therefore, the only factor to deal with is the additive noise from each of the sensors. Due to independence, having multiple sensors recording the same signal can be effective against additive white Gaussian noise, as mentioned in a previous section.

When processing the data by shifting in time and doing the LMMSE weighted sum, there was noticeable noise suppression when compared to all the individual sensor data. The signal to noise ratios were calculated and also showed improvements over all

the individual signals. This shows that the LMMSE weighted sum is effective in suppressing additive white Gaussian noise.

5.4 INTERFERENCE CASE

Not only can sensor arrays be effective against general AWGN, it can also be used to decrease the gain of interference from a direction not of interest. This technique is often called beamforming. Knowing the direction of the transmitted signal, the sensor array can delay and sum the received signals accordingly. By appropriately delaying, the summed signal from all the sensors can increase the gain of the signal from the corresponding direction. While a coherent signal in this direction is essentially being multiplied, interference and noise from other directions are being cancelled out. Therefore, the SNR of the desired signal is increased. Beamforming arrays are often adaptive to be used for direction of arrival applications as well.

Beamforming arrays are usually organized in a much more systematic way than our experimental setup. Often they are evenly spaced apart or setup in a grid. This makes the time delay calculations much easier and methodical such as for phased arrays. However, our setup still works as long as we know the position of each sensor relative to each other.

When processing the interference cases, all the data were realigned in time along with doing a weighted sum. For these cases, the definitely wasn't interference cancellation. The second source was still quite apparent in the processed output. However, it was a little smeared out. If listened closely, a bit of an echo is apparent. This is actually expected as the algorithm is actually just shifting the signal at different times, so multiple scaled instances of the signal are apparent. One could say the processed output is a slight improvement as the interference is smeared a bit. However, there is AWGN improvements, just like in the above case.

Canceling out a second source wasn't effective in this experiment because of the limited number of sensors being used. With a greater number of sensors in the array, interference cancellation would be much more effective.

5.5 MULTIPATHING CASE

A more complex channel to consider is one that produces multipathing. Multipathing is the result of the environment's geometry such that transmitted signals travel to the receiver by multiple different paths. The difference in the path lengths, attenuations, and number of bounces result in interference upon arrival at the receiver. This results in multiple arrivals of the signal which complicates the received signal. An example of a possible multipathing channel would be one within a totally enclosed box or room. With a transmitter on one side and a receiver on the other side, the received signal would contain multiple instances of the transmitted signal. This is due to the signal from the direct path along with paths that bounce off the six walls of the room. There may be higher order bounces as well.

We tried to mimic multipathing in as one of the experiment cases. We used a second source to produce the same signal as the main source, with a slight time delay. However, I believe simulating actual multipathing is difficult especially with just two sources. This is because each sensor will receive a signal at a time delay highly dependent on the geometry of the environment. This is difficult to reproduce.

When processing this case to try to cancel out the multipath, it was proven difficult. Much of the issues are the same as those with the interference case. Also, there was some trouble realigning the data due multiple instances of the same source signal. Once again, with a greater number of sensors in the array, I would suspect it to be much more effective.

7 FURTHER IMPROVEMENTS AND APPLICATIONS

There are a number of improvements that can be made to this project. When looking at the implemented encoder and decoder, some more components could have been used as described in Vosoughi and Scaglione's paper. One being the estimator. Another improvement that can be made is to better estimate h , or the channel. In this project, a simple attenuation and time delay model was used. Much more complex models could be used to better estimate the channel and perhaps get better results.

As for the experimental acoustic sensor array built, the one major improvement would be expanding the number of elements in the sensor array. If we could significantly improve the number of sensors, it would be much more effective in noise suppression. Also, the results from the interference and multipath cases would have been more noticeable with more sensors or data to work with. Another improvement would be implementing the encoder at each remote sensor. Therefore it would demonstrate an actual practical design instead of just experimental.

There are plenty of practical applications for this compression algorithm. As shown in this project, an acoustic sensor array using this algorithm is viable. This sensor array could be used for a number of purposes such as directional listening or beamforming, other surveillance applications, sonar, and of course noise suppression just to name a few. Using this algorithm, one can save much bandwidth in the communication within the array. Or perhaps the sensors don't communicate with a decoder immediately and instead store it on memory onsite. Using this algorithm, memory can also be saved if storage is needed. The compression algorithm can also be used for 2 dimensional processing such as imaging. As one can imagine, images take up much more bandwidth or memory, so compression would be extremely useful. A sensor array also allows lower quality sensors to be used. This due to the fact that noise can be more tolerable when using combined information. So there will be cost savings. Overall, the algorithm proves to be quite practical with many applications.

8 REFERENCES

- [1] A. Vosoughi and A. Scaglione, "Precoding and Decoding Paradigms for Distributed Data Compression," *Submitted to IEEE Tran. Sig. Proc.*, available at <http://crisp.ece.cornell.edu>
- [2] D. Slepian and J. K. Wolf, "Noiseless coding of correlated information sources," *IEEE Trans. Inform. Theory*, Vol. IT-19, pp. 471-480, July 1973.
- [3] S. Pradhan and K. Ramchandran, "Distributed source coding using syndroms (DISCUS): design and construction", *IEEE Trans. Inform. Theory*, Vol. 49, Issue 3, pp. 626-643, March 2003.
- [4] J.G. Proakis, *Digital Communications*. McGraw-Hill, New York, 2001.

9 APPENDIX

9.1 MATLAB CODE

Encoder

```
function [p,d]=encoder(x,wy,P)

load mindeltas2; %load minimum deltas
d=(wy/.1)*delta(P); %for set noise

%initializations
c=zeros(1,length(x));
p=zeros(1,length(x));
q=zeros(1,length(x));

%quantizing and cosets
c = round(x/((wy/.1)*delta(P)));
p = mod(c,P);
q = (c-p)/P;
```

Decoder

```
function xhat = decoder(p,y,wy,P)

load mindeltas2; %minimum deltas

%initializations
qhat=zeros(1,length(p));
chat=zeros(1,length(p));
xhat=zeros(1,length(p));

%minimum distance
qhat = round((y-p*(wy/.1)*delta(P))/(P*(wy/.1)*delta(P)));
chat =qhat*P+p;
xhat =chat*(wy/.1)*delta(P);
```

Minimum delta calculator

```
function mdelta=mindelta(P,noise)

x=randn(1,50000)'; %set gaussian signal
A = 1;
W = noise*randn(1,length(x))'; %noise
y = A*x+W;
n=1;

%numerical calculations
```

```

for P=P
    for i=0:.01:.2
        delta = i;
        c = round(x/delta);
        p = mod(c,P);
        q = (c-p)/P;
        qhat = round((y-p*delta)/(P*delta));
        k = qhat-q;
        MSE(P,n) = sum(k.^2.*(erfc((k-
0.5)*P*delta/noise/sqrt(2))/2-
erfc((k+0.5)*P*delta/noise/sqrt(2))/2))*P^2*delta^2+delta^2/12;
        n=n+1;
    end
    [y,index]=min(MSE); %minimum MSE
    mdelta(P) = (index-1)*.01; %minimum delat
end
figure
semilogy(MSE)

```

LMMSE weighted sum (for SNR)

```

function [output,C]=weightedsum(xhat,N,SNR,ax)

% calculate coefficients
for i=1:N
    C(i) = (1/ax(i)*SNR(i))/(sum(SNR)+1);
end

% combine inputs
output = zeros(1,length(xhat));
for i=1:N
    output=C(i)*xhat(i,:)+output;
end

```

Time shifts (cross correlation)

```

function [datashift, output]=tshift
load singlesource3
[x,fs]=wavread('test2');
x2=resample(x,10000,16000);

% range1=40000; %singlesource1
% range2=120000;

range1=10000; %singlesource2
range2=90000;

% range1=20000; %interfer2
% range2=90000;

```

```

% calculated DC offset
for i=1:4
    dc(i)=mean(data(200000:300000,i));
end
gauss1=data(range1:range2,1)-dc(1);
figure(1)

% cross correlations
for i=1:4
    [m(i),ind(i)]=max(xcorr(gauss1,data(range1:range2,i)-dc(i)));
    subplot(4,1,i)
    plot(xcorr(gauss1,data(range1:range2,i)-dc(i)))
end

% subplot(311)
% plot(xcorr(gauss1,data(range1:range2,2)-dc(2)))
% subplot(312)
% plot(xcorr(gauss1,data(range1:range2,3)-dc(3)))
% subplot(313)
% plot(xcorr(gauss1,data(range1:range2,4)-dc(4)))

%shifting data and offsetting DC
for i=1:4
    datashift(:,i)=[data(range2-range1+2-ind(i):end,i)-
dc(i);zeros(range2-range1+1-ind(i),1)];
end

% std calculations
for i=1:4
%     w(i)=std(data(165000:180000,i)); %interfer2
    w(i)=std(data(200000:300000,i));
    s(i)=std(datashift(range1:range2,i))-w(i);
end

%SNR calculations
SNR = s.^2./w.^2;
for i=1:4
    C(i) = (SNR(i))/(sum(SNR(1:4))+1);
end

%time delay and distance calculations
delay=(range2-range1+1-ind)/10000;
distance=delay*340*3.2808399;

% weighted sum
output = zeros(length(data),1);
for i=1:4
    output = C(i)*datashift(:,i)+output;
end
figure(2)
plot(datashift)

```

```

wo=std(output(200000:300000));
so=std(output(range1:range2))-wo;
SNRo=so^2/wo^2

```

Method 1 (MSE)

```

function MSE=method1
load singlesource3
[s,fs]=wavread('test2');
s=resample(s,10000,fs); %resample source

% standard dev calculations
sigmas=std(s);
s=s/sigmas;
sigmas=std(s);
for i=1:4
    dc(i)=mean(data(200000:300000,i));
    w(i)=std(data(200000:300000,i));
end

%normalization
for i=1:4
    A(i)=sqrt(((std(data2(:,i))))^2-w(i)^2)/sigmas^2);
    data2(:,i)=data2(:,i)/A(i);
end

% SNR calculations
SNR=sigmas^2*A.^2./w.^2;
wy=w(4)/A(4);
% wy=.22;

%shift data
data2=tshift;

% calculate MSE for different cosets
for P=1:256
    x(1,:)=data2(:,1)';
    x(2,:)=data2(:,2)';
    x(3,:)=data2(:,3)';
    y=data2(:,4)';
    for i=1:3
        [p,delta]=encoder(x(i,:),wy,P);
        xhat(i,:)=decoder(p,y,wy,P);
        % c=round(data2(:,1)/(m/(.5*(P-1)))); % only quantize
test
        % xhat(i,:)=c*(m/(.5*(P-1)));
    end
    [output,C]=weightedsum(xhat,3,SNR(1:3),[1 1 1]);
    % [p,delta]=encoder(x(1,:),wy,P);
    % xhat = decoder(p,y,wy,P);

```

```

%     MSE(P)=sum((x(1,:)-xhat).^2)/length(xhat);
MSE(P)=sum((s-output').^2)/length(s);
end
figure
loglog(MSE)
% figure
% plot(output)
% hold
% plot(s,'r')

```

Method 2 (MSE)

```

function MSE=method2
load singlesource3
[s,fs]=wavread('test2');
s=resample(s,10000,fs); %resmaple source

% standard dev calculations
sigmas=std(s);
s=s/sigmas;
sigmas=std(s);
for i=1:4
    dc(i)=mean(data(200000:300000,i));
    w(i)=std(data(200000:300000,i));
end

%shift data
data2=tshift;

%normalization
for i=1:4
    A(i)=sqrt(((std(data2(:,i))))^2-w(i)^2)/sigmas^2);
    data2(:,i)=data2(:,i)/A(i);
end

%SNR calculations
w=w./A;
w=[w(4) w(1:3)];
SNR=sigmas^2./w.^2;

%MSE calculations
for P=1:256
x(1,:)=data2(:,4)';
x(2,:)=data2(:,1)';
x(3,:)=data2(:,2)';
x(4,:)=data2(:,3)';
xbar(1,:)=x(1,:);

%iterative process for finding side information
for j=1:3

```

```

xhat=zeros(1,length(x));
C=[];
for i=1:j
    C(i) = SNR(i)/(sum(SNR(1:j))+1);
    xhat=C(i)*xbar(i,:)+xhat;
end
wxhat=1+1/SNR(j+1)+sum(C);
[p,delta]=encoder(x(j+1,:),wxhat,P);
xbar(j+1,:) = decoder(p,xhat,wxhat,P);
end

% for i=1:1
%     C(i) = SNR(i)/(sum(SNR(1:1))+1);
% end
% xhat=C(1)*xbar(1,:);
% wxhat=1+1/SNR(2)+C(1);
% [p,delta]=encoder(x(2,:),wxhat,P);
% xbar(2,:) = decoder(p,xhat,wxhat,P);
%
% for i=1:2
%     C(i) = SNR(i)/(sum(SNR(1:2))+1);
% end
% xhat=C(1)*xbar(1,:)+C(2)*xbar(2,:);
% wxhat=1+1/SNR(3)+(C(1)+C(2));
% [p,delta]=encoder(x(3,:),wxhat,P);
% xbar(3,:) = decoder(p,xhat,wxhat,P);
%
%
% for i=1:3
%     C(i) = SNR(i)/(sum(SNR(1:3))+1);
% end
% xhat=C(1)*xbar(1,:)+C(2)*xbar(2,:)+C(3)*xbar(3,:);
% wxhat=1+1/SNR(4)+(C(1)+C(2)+C(3));
% [p,delta]=encoder(x(4,:),wxhat,P);
% xbar(4,:) = decoder(p,xhat,wxhat,P);

[output,C]=weightedsum(xbar,4,SNR,[1 1 1 1]);
MSE(P)=sum((s-output').^2)/length(s);
end
% plot(output)

```