ABSTRACT

Title of thesis: USING A DISCRIMINATOR TO IMPROVE COMPRESSIVE SENSING EFFICIENCY

Kevin Hencke, MA, 2012

Thesis directed by: Dr. John Benedetto
Department of Math

Our work defines, implements, and evaluates a modification to a spectrum-based compression scheme for data streams coming from jet aircraft health-monitoring sensors. The modification consists of the addition of a discriminator which separates data streams into similar classes. We create and justify a simulation of a jet sensor network as a source for data streams. The data streams are compressed and decompressed under the new compression scheme and also under two old ones, and the reconstructions are evaluated for quality. The discriminator-based modification to the existing compression algorithm is found to yield better quality than the other two compression algorithms, at the cost of increased runtime.

USING A DISCRIMINATOR TO IMPROVE COMPRESSIVE SENSING EFFICIENCY

by

Kevin Hencke

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2012

Dedication

To Jess. You are my comfort, joy, and inspiration.

Acknowledgments

I have many people to be grateful to, and I hope those I forget will forgive me.

Many people have helped me on my journey to becoming a mathematician. I thank my earlier teachers, including Mrs. Smith and Mrs. Helms, for teaching me to love math. I thank Dr. Todd Moon for giving me something to do with math by introducing me to signal processing. I thank Dr. Mike Boyle for helping me continue my journey by directing me to the Norbert Wiener Center. I thank Dr. John Benedetto and Dr. Wojciech Czaja for mentoring, advising, and supporting me during these past few years with the NWC, and allowing me to continue my studies as a graduate student.

I thank Dr. Mike Dellomo for bringing me a fascinating project to work on, for letting me write my master's thesis about it, and for many highly informative and entertaining meetings spent discussing our algorithm (and all manner of other things). I also thank his associates Mr. Tej Phool and Dr. Gang Qu; thank you for sharing your project with me.

For things less mathematical but just as important, I thank my friends and family, especially Jess, for their loving support.

I also thank Ms. Dorothea Brosius for maintaining a tremendously helpful LaTeX thesis template, at the following site:

`http://www.ireap.umd.edu/ireap/theses/LatexThesisTemplate/`

Table of Contents

List of Figures

Chapter 1

Introduction

The task of actively monitoring the health of running aircraft has obvious worth, and has been the subject of much prior work. Hardware systems employ sensors to monitor various conditions, including vibration and oil temperature, to detect damage as it happens and warn pilots of dangerous situations. As technology has advanced, though, we have become increasingly good at gathering data, to the point where it becomes cumbersome to handle it all without some type of data compression.

In this paper we detail a variation of a data compression scheme for transmitting sensor feeds to a central aircraft health monitoring system. The compression is based on projection of the magnitude frequency spectrum of the data feeds onto a basis, which is obtained from a number of training clippings by principal component analysis. We introduce a new variant of the scheme by applying a 'discriminator,' which (for now) means manually dividing the sensors into similar classes, and performing the compression algorithm by class.

We begin by explaining and justifying a simulation of aircraft health sensor data feeds, from which we draw our data to be compressed. We then run a number of trials on different data sets, applying various compression ratios and grouping the sensors in three different ways: all in one group, grouped by true class, and processed individually. This entire experiment is intended as preliminary work, with a future experiment planned that will introduce highly parallel computing and an automated, algorithmic discriminator.

Chapter 2

Theory

2.1 Background Information on Aircraft

In our experiment we model an aircraft whose components are vibrating under the effects of normal cruising flight. To justify our simulation choices, we will make some comments on the structure of jet engines, and the nature of vibrations caused by machinery and airflow.

2.1.1 Structure of Jet Engines

A jet engine is a combustion-driven propulsion system that is commonly used to propel aircraft. A diagram of two jet engines is provided in Figure 2.1, which is drawn from [12]. The diagram pictures the engines in cutaway view, with the cutting plane parallel to the axis of rotation. The inner turbines are shaded dark and the outer turbines (including their intake fans) are shaded more lightly; the inner and outer turbines of each engine spin freely and are not coupled to each other.

In the diagram, air flows from the left of the engine to the right. Jet engines have four stages of combustion, all of which happen simultaneously at different areas in the engine. Each successive stage occurs further back in the engine:

1. The intake fan and compressor blades at the front of the engine spin, pulling air in and forcing it into a narrowing duct, pressurizing it.

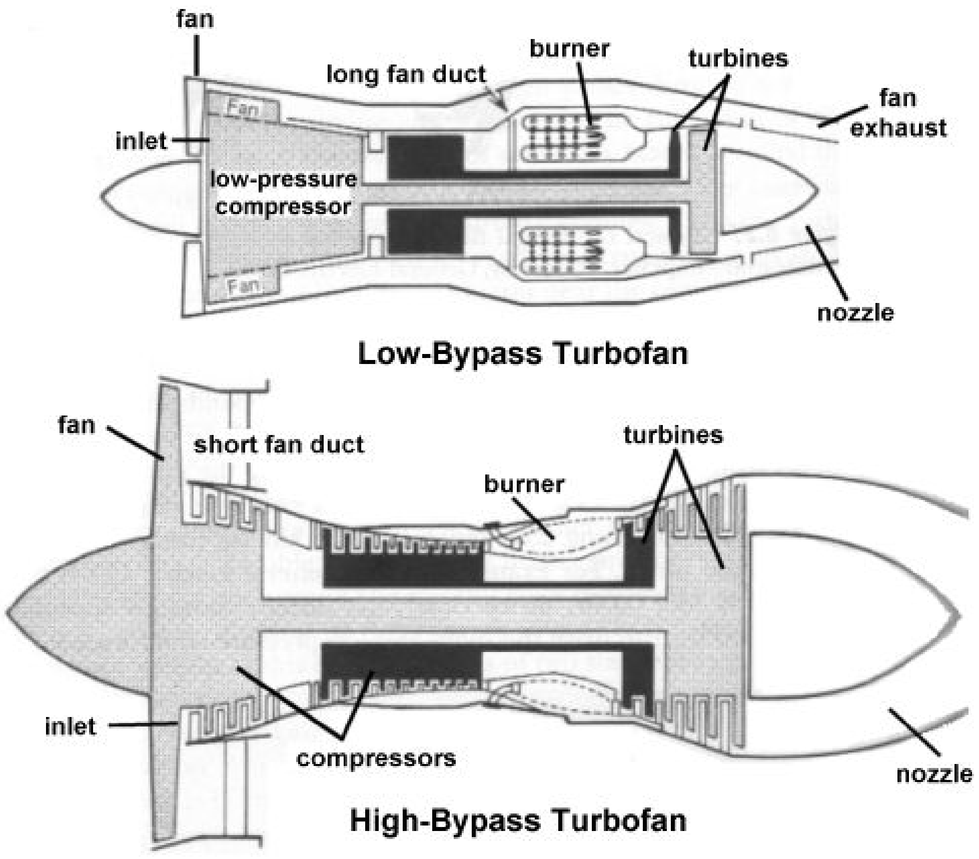

Figure 2.1: Two examples of jet engines.

2. In the burners, jet fuel is injected into the compressed air, and ignited. The combustion increases the pressure of the air.

3. The pressurized exhaust blows backwards through the engine, powering the drive turbines and providing torque to the intake fan and compressor blades.

4. The depleted, pressurized exhaust exits the jet engine, providing thrust to the aircraft and making room for fresh air.

2.1.2 Vibration in Aircraft

One detection method for identifying change or damage to aircraft uses Fourier analysis. A change in the magnitude of the spectra of vibration on an aircraft indicates some change to the mechanical component of the aircraft associated with that spectral component. Machinery is generally designed to minimize superfluous vibration and heat, as extra vibration and heat constitute an unintended loss of useful energy, and contribute to wear.

Therefore, it is reasonable to associate increased vibration, noise, and heat with damage to machinery. Vibration sensors have established utility in detecting and predicting faults in machinery. [7] It is also useful to consider the spectra of vibrations, since vibrations produced under stable operation come from components that exhibit periodic behavior. Past work has made clear the usefulness of Fourier analysis in fault detection. [2] [11] The magnitude of FFTs is what concerns us the most. Phase variation is less important to us because it does not contribute to power. In this work we only study magnitudes of FFTs.

Each rotational component of an engine under stable operation will produce a periodic vibration pattern as the component makes successive cycles and returns to its starting point. Suppose there exists a driveshaft in an engine that rotates at τ Hz, and no other component spins at that speed. Since the driveshaft's vibration function is τ-periodic, the FFT of this vibration will consist of harmonics of τ and will be easily identified among the engine's vibrational spectra. If we are monitoring the vibrational spectrum of this engine and we see a sudden increase in the power domain corresponding to τ Hz and its harmonics, it is likely an indication that something has changed with the functioning of the driveshaft.

In a similar way, we assert that damage to the fuselage of a flying aircraft may be detected as a change in vibrational spectra. An airplane fuselage in flight will vibrate with spectra well-approximated by pink noise; a precedent for this assumption is established in [5]. A stable, rigid body subject to strong airflow will vibrate in a consistent way, and a dramatic change in the character of vibrations may again be taken to mean that some change, possibly including damage to the fusleage, has occurred.

Depending on the situation, a change in vibrational spectra may be an indication of new damage that threatens to cause catastrophic failure [4]. In this case, monitoring spectra can help to prevent or minimize damage to equipment, or even prevent injury or death. It may even be the case that by analysis of the spectra and comparison to previous records of healthy spectra, the damaged component may be identified specifically, allowing not only prevention of accidents but also ease of maintenance. [4] [1]

A τ-periodic signal has a spectrum dominated by a sine of frequency τ, plus harmonics (multiples) of τ. For this reason we model vibrations of single engine components as simple sine waves, and ignore harmonics. This is simplistic, and it would be informative to obtain and study real vibrational data from jet engines. For the purpose of this paper, the vibration of one engine is modeled as the sum of vibrations of turbine shafts, blades, gears, and a zero-mean white noise term:

$$v^i(t) \quad = \quad N^i(t) + \sum_{j=1}^{T} \left[\begin{array}{c} p^S_{(i,j)} sin(2\pi\tau_{(i,j)}t) + \\ p^B_{(i,j)} sin(2\pi\tau_{(i,j)}b_{(i,j)}t) + \\ \sum_{k=1}^{G_{(i,j)}} p^G_{(i,j,k)} sin(2\pi\tau_{(i,j)}g_{(i,j,k)}t) \end{array} \right] \qquad (2.1)$$

We will talk about this summation more specifically in 3.2.2.

We note that vibrations travel throughout machinery; the vibrations produced by one engine will not remain confined to that engine, and likewise any wind-driven vibration of the fuselage will also travel to the engines, under the effects of some attenuation. Accordingly, we assert that when simulating a network of sensors and noise sources on an aircraft, a weighted graph, whose edges specify attenuation factors, provides a good model of the movement of vibrations. We will discuss this more specifically in 3.2.4.

Our algorithm is a compression scheme intended for use with a spectral-based fault detection system, as it bases reconstruction on the shape of typical spectra. It does not itself include a fault detection system. We would welcome an augmentation of this work with such a detection system, and are planning to construct such an experiment ourselves.

2.2 Discussion of Compression

2.2.1 Background

In any system where data must be transmitted but bandwidth is scarce, it may be desirable to use some form of data compression. There are many types of compression, and they generally exploit redundancy in the data. As an example, data with long sequences of identical values may use run-length encoding. Huffman coding, which exploits commonly occurring patterns, is another possibility; both it and run-length coding are lossless, but such codes are not our only options. [3] [10]

We may certainly restrict ourselves to lossless encoding, but another option is to use lossy compression based on discarding unimportant or insignificant information. Computationally, this frequently means exploiting sparsity in the data, and possibly applying a transformation first to obtain the desired sparsity. Sparse data is easy to compress because by definition it contains large numbers of zeros and is therefore high in redundancy. If we understand our data well, we may be able to choose an effective transformation in order to bring many of the data coefficients to zero, or within some threshhold of zero. We may threshhold the data to obtain high sparsity at minimal loss of accuracy or meaning. [10]

Two examples of types of lossy transformations useful in compression are Fourier-based transforms and wavelet transforms. The former exploits periodicity by projecting onto a basis of trig functions. In contrast, the latter exploits localized discontinuities by projecting onto a basis that separates matrices into coefficients describing local averages and local directional changes. Data that is highly periodic (such as an image containing a regular, repeating pattern; for example a brick wall) is handled well by discrete Fourier transforms, while data with highly localized discontinuity, such as cartoon images, is handled well by wavelet transforms. [10]

In both the Fourier and wavelet case stated above, we assume that our data, which lies in some space S, is essentially a linear combination of a relatively small number of basis vectors from a carefully chosen basis, plus insignificant amounts of noise. Another type of compression along these same lines relies on an assumption that the data, which may be high-dimensional, takes the form of points on a low-dimensional linear manifold, plus a small amount of insignificant random noise. If this is the case, then we may make a guess as to the form of the manifold and/or a basis set for the manifold. If we manage to guess well, we may compress any high-dimensional data point by a smaller number of parameters by projecting it onto the basis set. In this case, we reconstruct it as follows:

$$x^* = \sum_{b_i \in B} b_i \langle x, b_i \rangle \text{ for } x \in span(B) \subset S$$

For $x \in S$ that lie close to $span(B)$, this is a good approximation, and if $|B| << dim(S)$, we may achieve a high compression rate with little effort or loss of data, by choosing B cleverly. This is the goal of compressive sensing.

2.2.2 Redundancy of Jet Vibrational Data

We assume that our simulated aircraft's vibration sensors are all picking up linear combinations of the same five sources: the four engines and the fuselage noise. Our data, then, is close to 5-dimensional in magnitude spectrum, plus some noise. We believe that the magnitude spectra of each sensor lie near the subspace that is spanned by the average magnitude spectra of the five vibrational signal sources. Principal component analysis is well-established as a technique for undoing linear mixing, so we attempt to use it to recover the signal sources from the linear combinations therein that comprise the sensor clippings, and use several of the first resulting principal components as our basis. A detailed treatment of PCA is set forth in [9].

In this way, by choosing and projecting onto a basis of size 100 or less, a vector of length 2^{13} is reduced to 100 coefficients or fewer. It is important to note that we must store the compression basis we are using, which itself contains many vectors of length 2^{13}, on both sides of the restrictive channel. However this is a one-time cost, and is easier than continually passing new vectors of length 2^{13} through the channel. We take several training clippings to form our basis, and then we only transmit coefficients from then on.

Our innovation consists of sorting the sensor feeds. Rather than compressing the temperature, fuselage, and engine sensors all together, or separating them entirely, we group them manually into their three natural classes. In this way we hope to observe that a better, more fitting basis may be created for each class. In running trials, we look for an improvement in efficiency or performance given by the different compression schemes made possible by sorting the data streams by class before determining a compression scheme.

All analysis and compression is done in the spectral domain. The time series vibrational data is real-valued and represents pressure with respect to time, as in the usual case of auditory or vibrational data. A fast Fourier Transform is performed on each clipping, and the absolute value of the resulting frequency-domain data is taken. The result is a nonnegative, real-valued spectrum. In taking the absolute value we discard the phase information of the clippings; as discussed previously, we do not consider phase data in this project.

Chapter 3

Implementation

3.1 Note on Discretization

We noted that the highest-frequency sine arising from our experiment was that corresponding to the blade frequency of an inner turbine, which had an average frequency of $3.3786 \cdot 10^3$. [13] With this in mind, we estimate that the Nyquist frequency [8] is about 8000, and as such we chose to simulate a sampling rate of $2^{13} = 8192$ samples per second. We ran the experiment for 64 seconds, for a total of $N = 2^{19}$ discrete moments in time. Thus the previously discussed matrix S has 12 columns and 2^{19} rows. For purposes of compression, we will divide the sensor streams into clippings of length 8192.

All of our computations are done in MATLAB, and the PCA is done based on the built-in SVD algorithm. The source code used to generate the data, compress and decompress, and evaluate performance are set forth in Appendix B.

3.2 Generation of Dataset

3.2.1 Plane structure overview

We model our plane as having five engine sources: four unique engines, and the wind rushing over the fuselage. We simulate there being one vibration sensor on each engine and four along the length of the fuselage. We also simulate temperature sensors on each engine, for a total of 12 sensors.

3.2.2 Engine Vibration

As previously discussed in 2.1.2, we model engine vibration as a sum of sines, with frequencies and magnitudes chosen to represent the shafts, blades, and gears of an engine, as well as additive white noise. When viewed as a function, the vibrational signal v^i coming from the ith engine takes the form of the sum of the vibration of shafts, blades, gears, and white noise, or:

$$v^i(t) \quad = \quad N^i(t) + \sum_{j=1}^{T} \left[\begin{array}{c} p^S_{(i,j)} sin(2\pi\tau_{(i,j)}t) + \\ p^B_{(i,j)} sin(2\pi\tau_{(i,j)}b_{(i,j)}t) + \\ \sum_{k=1}^{G_{(i,j)}} p^G_{(i,j,k)} sin(2\pi\tau_{(i,j)}g_{(i,j,k)}t) \end{array} \right] \tag{3.1}$$

Here j indexes each of the T turbines. The rotational speed of the jth turbine shaft in the ith engine, measured in Hz, is denoted $\tau_{(i,j)}$. The various p coefficients parameterize the volume of each component. The constant $b_{(i,j)}$ is defined as the number of blades of the jth turbine of the ith engine. $G_{(i,j)}$ is the number of gears engaged with a turbine, and $g_{(i,j,k)}$ are the corresponding gear ratios. Overall, we have a sum of a shaft sine, a blade sine, and gear sines, and we sum this value across all turbines of the engine. The function $N^i(t)$ is a random variable with Gaussian distribution and some given scalar constant establishing magnitude; this term adds white noise to the engine.

In our experiment, we use four identical engines, with slight variations introduced into the rotational speeds of the turbines and volumes of the gears. We base the specifications of our simulated engines on [13]. For each engine, we model two turbines (outer and inner) in each engine, with 20 blades on the outer turbine and 21 on the inner. We let the outer turbine's rotational speed be 2069 rpms, and we set the inner turbine to spin at 9653 rpms. We then multiply each turbine's spin speed by an iid random coefficient X, uniformly distributed on $[.95, 1.05]$, to introduce 5% random variation. We then converted rpms to Hz to get

$$\tau_{i,1} = 2069X/60, \qquad \tau_{i,2} = 9653X/60$$

We set the volume scaling parameters $p^S_{(i,j)} = p^B_{(i,j)} = 1$. It would be interesting to see the experiment repeated with other values for these parameters.

For each engine we model one gear engaging the outer turbine with gear ratio 5/13, and two gears engaging the inner turbine with ratios 7/4 and 1/2. We let each coefficient $p_{(i,j,k)}$ have mean $10^{-3/20}$ and multiply by an iid random coefficient, uniformly distributed on $[.95, 1.05]$, as we did previously with turbine shaft spin speeds. Thus

$$p_{(i,j,k)} = 10^{-3/20}X \quad \forall (i, j, k) \tag{3.2}$$

$$g_{(i,1,1)} = 5/13 \tag{3.3}$$

$$g_{(i,2,1)} = 7/4 \tag{3.4}$$

$$g_{(i,2,2)} = 1/2 \tag{3.5}$$

Now that we have defined the vibration function for each engine, we discretize it as $v^i_n = v^i(2^{-13}n)$, defined in terms of the nth moment in time and the sampling rate 2^{13}. We may now take the FFT of the vector $\hat{v}^i$, which consists of every v^i_n

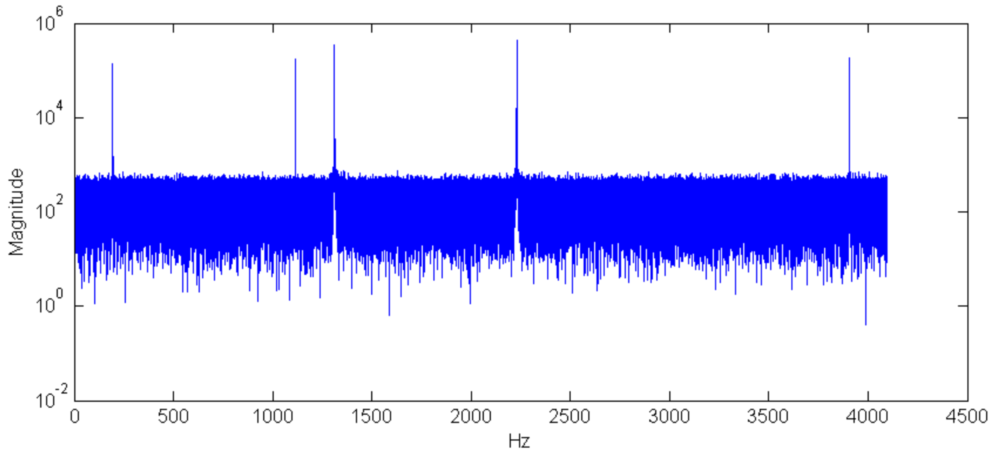

Figure 3.1: Magnitude of engine vibration spectra

gathered together. The magnitude of the FFT of one such $\hat{v}^i$, the entire vibrational signal from one engine, is displayed in Figure 3.1.

3.2.3 Fuselage Vibration

In our experiment, fuselage vibration is modeled as pink noise, as is cockpit vibration in [5]. We create our pink noise in terms of its FFT, with $1/f$ magnitude and phase randomized uniformly over $[0, 2\pi]$. We must be careful in our definition of this noise because we want our time-domain signal to be entirely real, so we must ensure that the FFT of our pink noise is symmetric. For some background information on types of noise, see Appendix A.

We start by defining the vector x with $x_j = 1/j$ for $j = \{1 \ldots (N/2) - 1\}$. We then remove the last term from x and reverse the order of the remaining coefficients to define r. Thus the vector $[x, r]$ is palindromic. Finally, we define the concatenated vector $m = [0, x, r]$; this is the magnitude of the FFT of the fuselage vibration.

We define the phase by a similar process, letting y be defined as $(N/2) - 1$ realizations of a random variable U that is uniformly distributed over $[0, 2\pi)$. We then exponentiate to create z, letting $z_j = e^{iy_j}$, where i is used as the imaginary unit in this case. We now let s be formed from z as r was formed from x before; we truncate the last term of z and reverse the order to get s. Now we define the vector $p = [1, z, s]$; this is the phase of the fuselage vibration's FFT.

We form the vector $\hat{f}$ by the following rule:

$$\hat{f}_j = 1000 m_j p_j$$

We use the constant 1000 to adjust the overall power of the fuselage noise. We now take the inverse FFT of $\hat{f}$ to obtain the vector of fuselage vibrational signals, f. Because we have defined the magintude and phase of the FFT symmetrically, the resulting f is real-valued, as desired.

3.2.4 Propagation of Signals

Figure 3.2 describes the propogation of vibration through the aircraft in our simulation. One engine and one sensor are located at each of points E1-E4, while the fuselage sensors are located at points F1-F4. The blank point in the middle of the graph contains no sensors and produces no noise and only serves as a conduit for vibration. The engines at points E1-E4 produce vibrations which are attenuated by the multiplicative constants $\{a, b, c, d\} \in (0, 1)$ as they travel along each node of the graph. As an example, the sensor at F1 picks up the vibrations created by engine E4, with an overall attenuation factor of $a \cdot b \cdot c \cdot c = abc^2$.

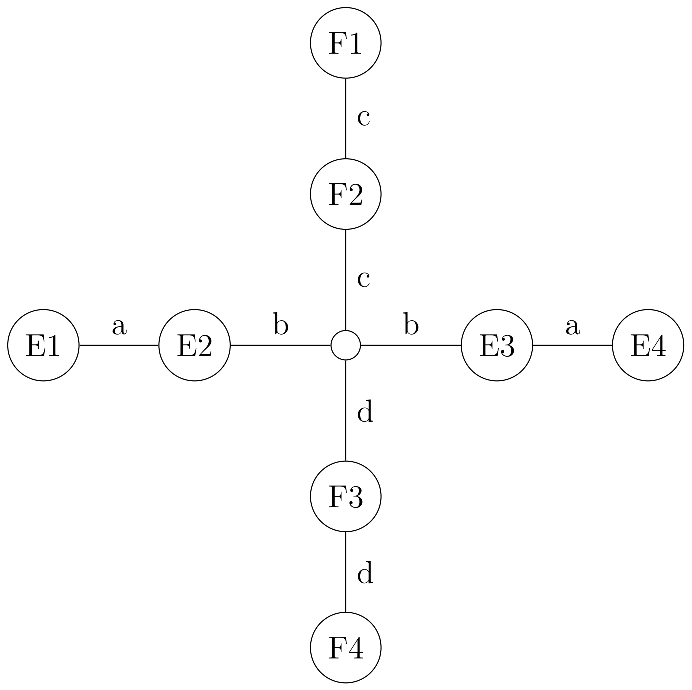

Figure 3.2: Weighted Graph depicting propagation of vibration

In our experiment we let $(a, b, c, d) = (10^{-10/20}, 10^{-16/20}, 10^{-18/20}, 10^{-13/20})$. Vibration are simulated as traveling instantaneously; this may be unrealistic, and it would be interesting to see the experiment repeated with delays added at each node of the graph.

3.2.5 Placement of Sensors

Our experiments use twelve sensors, labeled S1–S12. Sensors S1–S8 are vibrational sensors and S9–S12 are temperature sensors. Referring again to Figure 3.2, S1–S4 are located at points E1–E4, while S5–S8 are located at F1–F4. The temperature sensors S9–S12 are also located at E1–E4. Temperature is a constant value unique to each engine, which does not propagate. The sensors S9–S12 read a single, constant value throughout the experiment.

3.2.6 Signal Mixing

We may consider the realizations of sensors S1–S12 at timestep n as the values $\{s_n^1, \ldots, s_n^{12}\}$. Similarly, we label the engine vibrations and temperatures produced at timestep n by the engines at E1–E4 as $\{v_n^1, \ldots, v_n^4\}$ and $\{t_n^1, \ldots, t_n^4\}$, and the fuselage vibration as f_n.

The temperature sensors S9–S12 are always exactly equal to the temperatures of the corresponding engines and there is no mixing, so we will not discuss them further. However, the values read by the vibration sensors at time n are determined by the fuselage and engine vibrations by a matrix multiplication, for a mixing matrix M defined as follows in equation (3.6):

$$\begin{bmatrix} s_n^1 \\ \vdots \\ s_n^8 \end{bmatrix} = \begin{bmatrix} 1 & a & ab^2 & a^2b^2 & abc^2 \\ a & 1 & b^2 & ab^2 & bc^2 \\ ab^2 & b^2 & 1 & a & bc^2 \\ a^2b & a^2b & ab & 1 & abc^2 \\ abc^2 & bc^2 & bc^2 & abc^2 & 1 \\ abc & bc & bc & abc & c \\ abd & bd & bd & abd & cd \\ abd^2 & bd^2 & bd^2 & abd^2 & cd^2 \end{bmatrix} \begin{bmatrix} v_n^1 \\ \vdots \\ v_n^4 \\ f_n \end{bmatrix} \tag{3.6}$$

We anticipate further work on this project. As such, since running the experiments whose results we include in this paper, we have made minor adjustments to our weighted graph and our mixing matrix. We have represented these revised forms in Equation 3.6 and Figure 3.2. The general form and function of these are the same

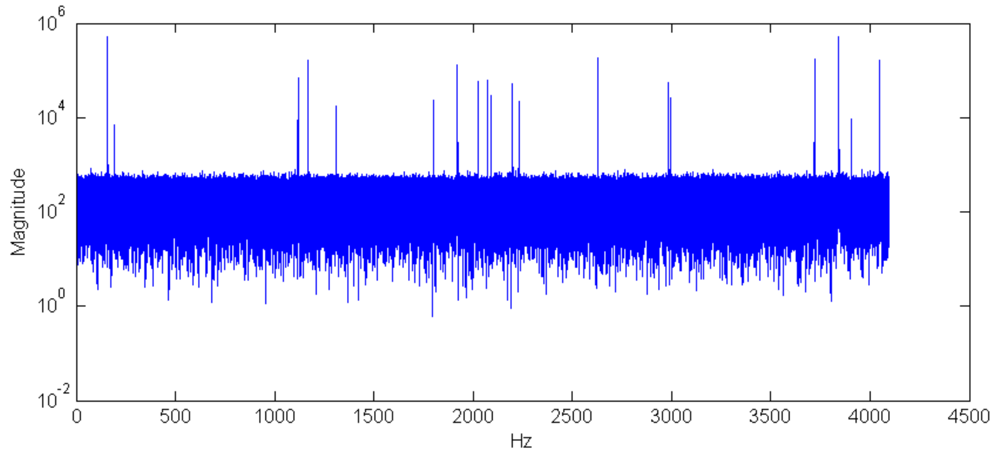

Figure 3.3: Magnitude of spectra of one inner engine sensor

as the graph and mixing matrix used in our experiments.

We stack the vectors $\bar{s}_n = [s_n^1 \dots s_n^{12}]$ vertically, forming the overall sensor data matrix S set forth in equation (3.7).

$$S = \begin{bmatrix} \bar{s}_1 \\ \vdots \\ \bar{s}_N \end{bmatrix} = \begin{bmatrix} s_1^1 & \dots & s_1^{12} \\ \vdots & & \vdots \\ s_N^1 & \dots & s_N^{12} \end{bmatrix} \tag{3.7}$$

Examples of the magnitudes of the spectra of an inner engine sensor, outer engine sensor, and a fuselage sensor are provided in figures 3.3, 3.4, and 3.5. The stronger low-frequency signal provided by the fuselage vibrations is visible in the lower frequency range of 3.5.

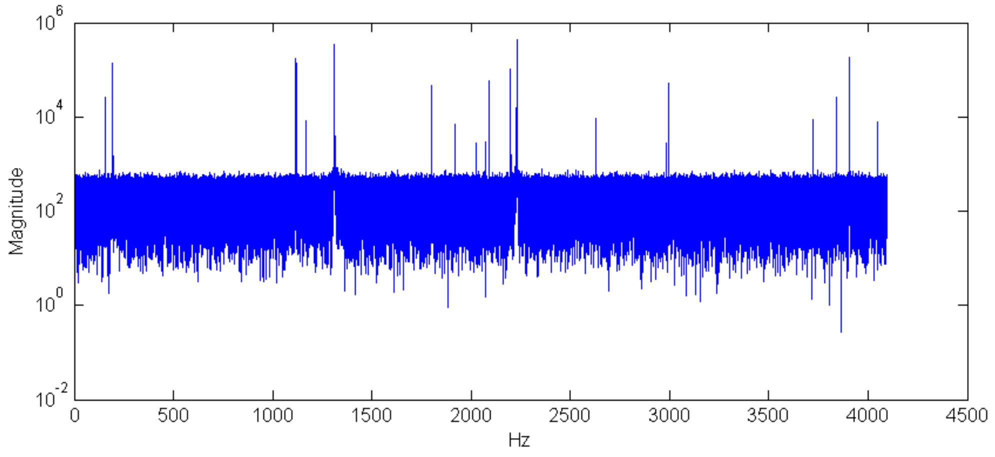

Figure 3.4: Magnitude of spectra of one outer engine sensor

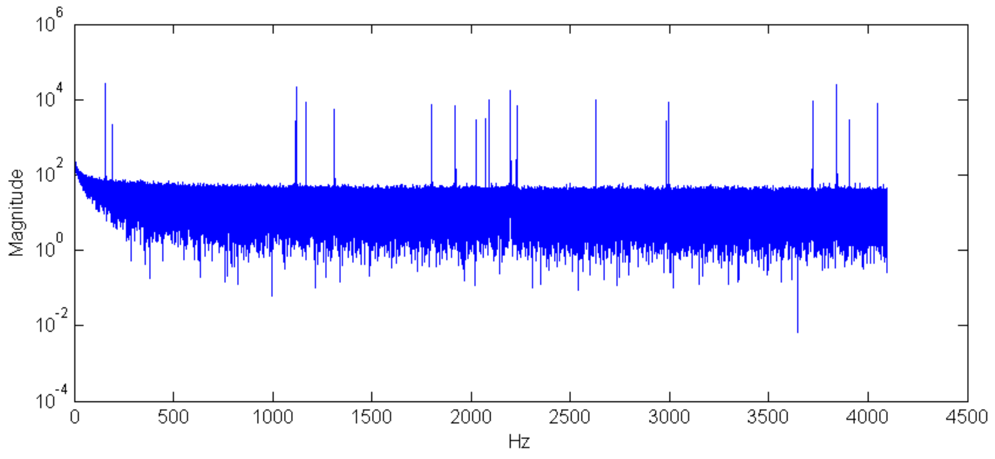

Figure 3.5: Magnitude of spectra of one fuselage sensor

3.3 Design of Compression Algorithm

3.3.1 Overview

Our compression strategy is to divide each sensor feed into clippings and take the FFTs of these, discarding the phase and keeping only the magnitude. We then subtract an previously computed average vector, the "compression mean," and project the result onto a "compression basis" and then transmit the resulting basis coefficients. We decompress by reversing this process; note that phase is not reconstructed.

3.3.2 Initialization of Compression Strategy

Before we can perform compression, we must build the compression mean and compression basis. These are computed from a number of training clippings taken from the sensor data matrix S, by the following procedure:

1. Establish a desired number y of basis vectors; this establishes the compression ratio as $8192/y$
2. Choose a sensor index set $\Lambda \subset \{1, \ldots, 12\}$, of size $|\Lambda| = x$
3. Form the matrix $S_\Lambda = [s^{\Lambda_1}, \ldots, s^{\Lambda_x}]$ drawn from the total sensor recordings
4. Establish a desired number z of training clippings from the sensors
5. Choose a training index set $\Gamma \subset \{1, \ldots, N - 8192\}$, of size $k = \lceil z/x \rceil$

6. Form the training matrix T of dimensions $8192 \times kx$, as $T = [T_1, \ldots, T_k]$ with

$$T_i = \begin{bmatrix} S_\Lambda(\Gamma_i, 1) & \ldots & S_\Lambda(\Gamma_i, x) \\ \vdots & & \vdots \\ S_\Lambda(\Gamma_i + 8192, 1) & \ldots & S_\Lambda(\Gamma_i + 8192, x) \end{bmatrix}$$

7. Take the FFT of the columns of T to obtain $\hat{T}$

8. Take the absolute value of all values of $\hat{T}$, discarding all phase information and retaining only magnitude information. Label this U

9. Take the row-wise means of U to obtain a single average column vector μ of size 8192×1

10. Subtract μ from every column of U to obtain the matrix V

11. Perform PCA on the columns of V to get a basis B of column vectors of size 8192×8192

12. Truncate the basis B to be size $8192 \times y$

This yields the compression mean μ and the compression basis B.

3.3.3 Performing Compression and Reconstruction

Given a clipping a of size 8192×1 taken from one of the sensors of S_Λ, to compress it we define $b = |FFT[a]| - \mu$, and then project b onto the basis B to form the coefficient vector $c = B^T b$. c has size $y \times 1$, for a compression ratio of $8192/y$. To reconstruct, we compute $a^* = IFFT[Bc + \mu]$.

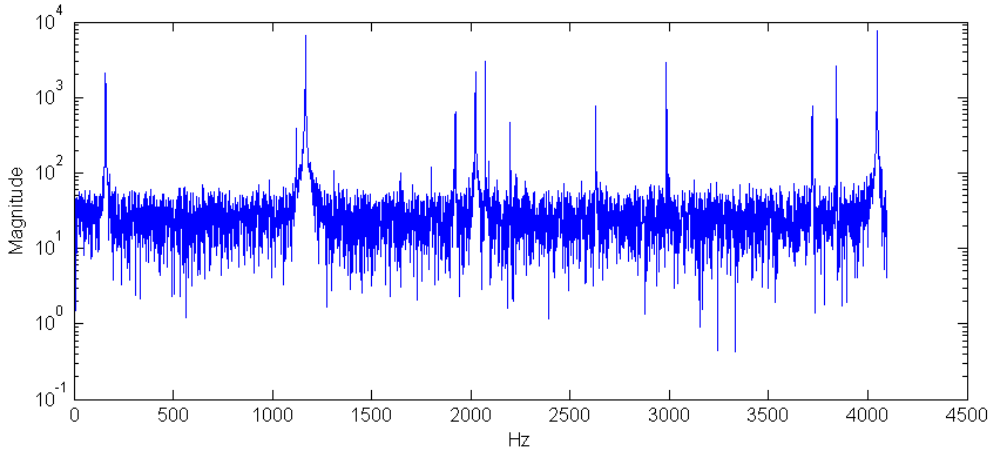

Figure 3.6: Spectra of one engine sensor clipping

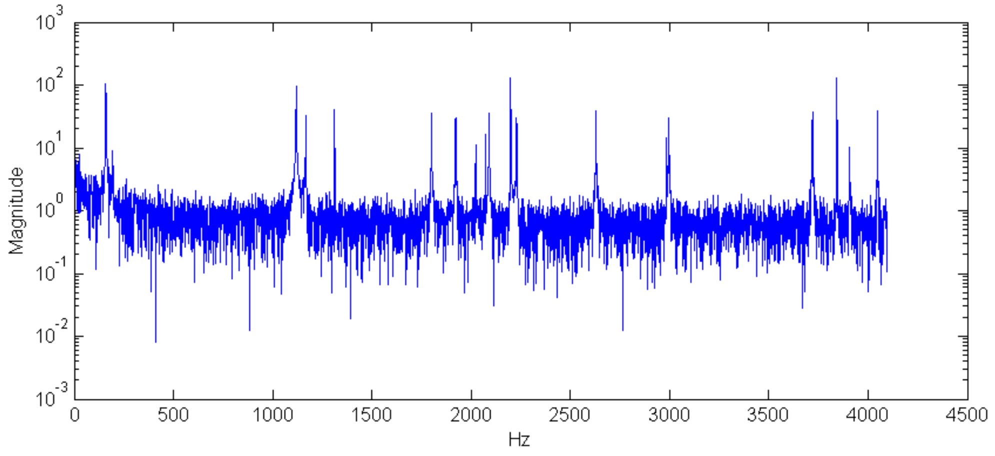

Figure 3.7: Spectra of one fuselage sensor clipping

An example of the magnitude spectrum of a 8192×1 engine sensor clipping is displayed in Figure 3.6, and the magnitude spectrum of a clipping of a fuselage sensor is pictured in Figure 3.7. Frequency spikes, caused by the powerful sines contributed by engines, are visible in each plot. The white noise is also visible, and the $1/f$ magnitude pink noise of the fuselage sensor is apparent in the low-frequency range of Figure 3.7.

3.3.4 Applying Discriminator

Work has already been done in this area using a full sensor index set $\Lambda = \{1, \ldots, 12\}$ or the degenerate case of $\Lambda = k \in \{1, \ldots, 12\}$; this corresponds to compressing every sensor at once by the same basis, or every sensor individually. The novel idea applied in our work is the discrimination of sensors into three different classes, corresponding to $\Lambda_1 = \{1, \ldots, 4\}$, $\Lambda_2 = \{5, \ldots, 8\}$, and $\Lambda_3 = \{9, \ldots, 12\}$, and applying the compression algorithm set forth above to each class separately. This yields three compression bases and three compression means. We run reconstruction performance comparisons on the cases of the undiscriminated case, the three-class discriminated case, and the signal-by-signal, using the same *total* number of basis vectors in each case, but dividing them between classes by hand.

In the three-class case we discriminate by hand with a priori knowlege, but we intend to continue our work with an automated discriminator which requires no a priori knowledge and which can separate the sensors into classes automatically, as well as allocate basis vectors to each class automatically.

3.4 Results

3.4.1 Note on Temperature Sensors

Since we simulated the temperature sensors as having constant values, they are computationally uninteresting. When the temperature sensors are treated as a separate class, the compression mean alone is sufficient to perfectly reconstruct

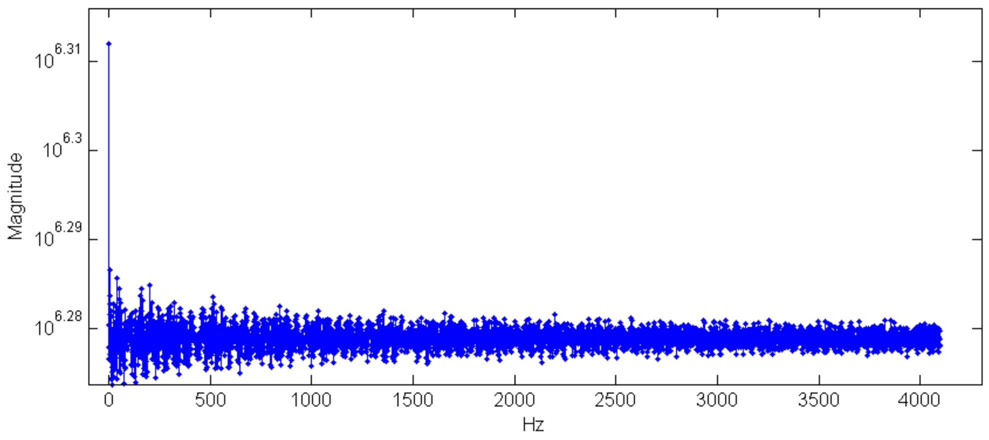

Figure 3.8: Compression mean in unsorted case

every clipping.

3.4.2 Compression Means and Eigenvalues

For an example case, a dataset was generated and the compression strategy was initialized for the undiscriminated and three-class discriminated case. 80 basis vectors total were allocated in both cases. In the discriminated case, the engine sensors were allocated 54 basis vectors and the fuselage sensors were given 25, while the temperature sensors received only a single basis vector.

By allocating the same total number of basis vectors in both cases, we hoped to closely compare the performance of the discriminated and undiscriminated compressor. It is important to note that the undiscriminated case has only one compression mean and the discriminated case has 3, but we feel this may be overlooked as the compression mean is a one-time cost. Once the mean/s are stored, the same number of compressed coefficients are transmitted for every clipping.

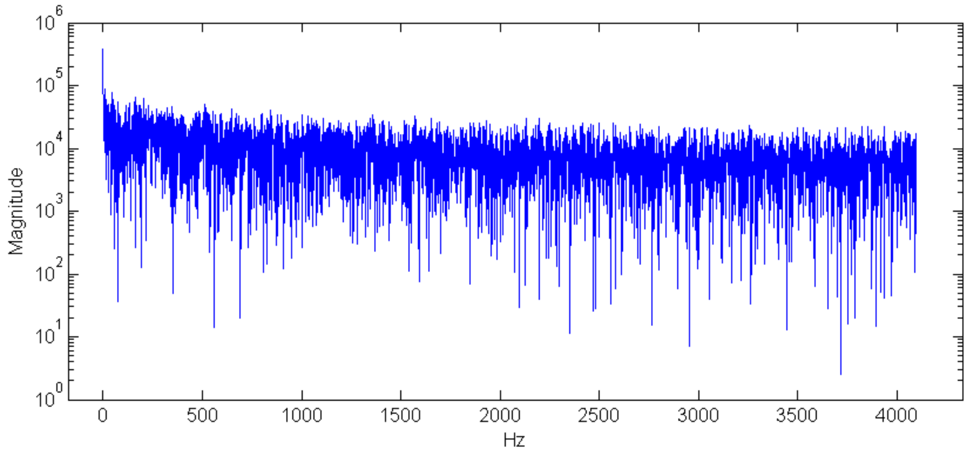

Figure 3.9: Compression mean of only engine sensors

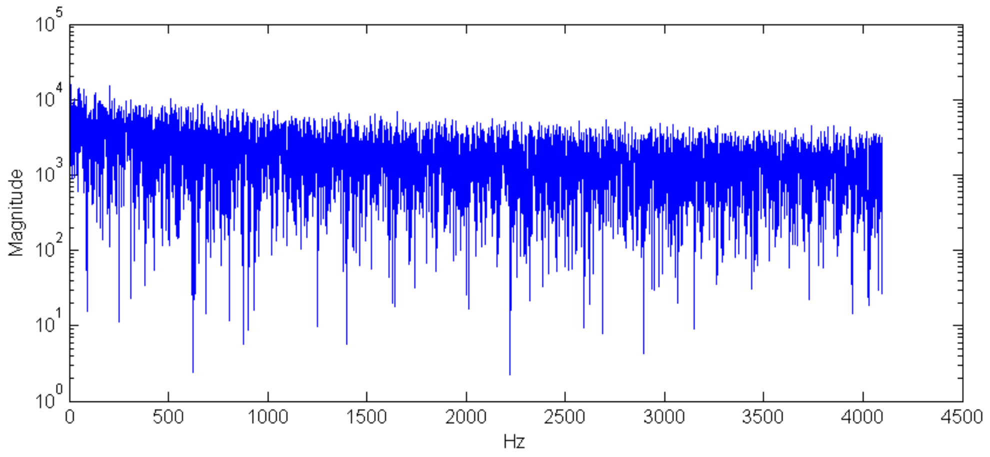

Figure 3.10: Compression mean of only fuselage sensors

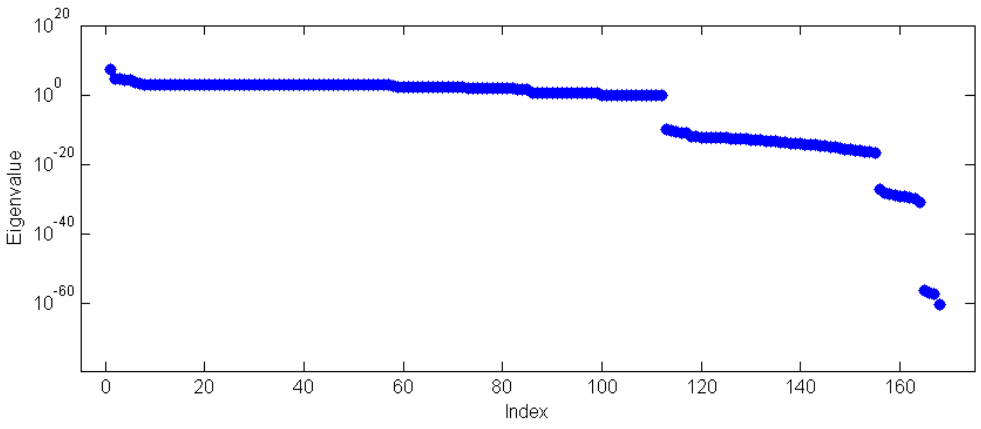

Figure 3.11: Plot of eigenvalues of all clippings

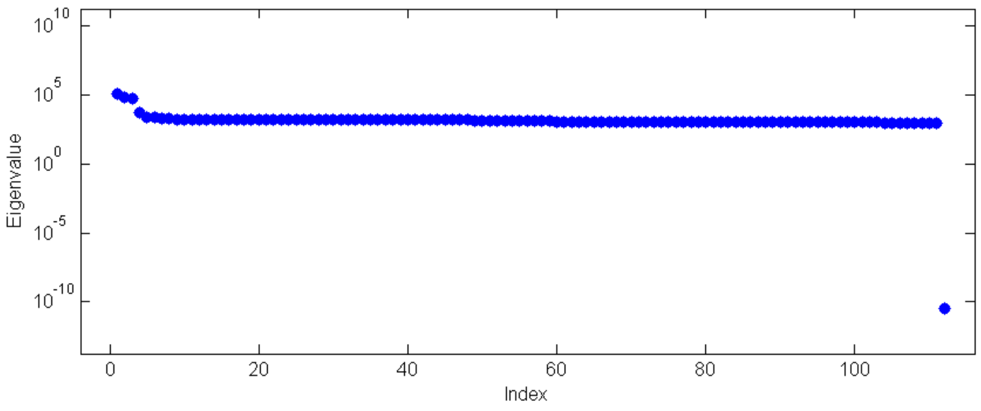

Figure 3.12: Plot of eigenvalues of engine sensor clippings

The compression means are pictured in Figures 3.8, 3.9, and 3.10. The compression mean in the undiscriminated case, shown in Figure 3.8, is plotted with marked points to accentuate the zeroth FFT coefficient. This spike corresponds to the constant, high-magnitude of the temperature sensors. This feature is not present in any vibrational sensor and we expect to improve our performance by discriminating out the temperature sensors and removing this irrelevant feature from the compression of vibrational sensors.

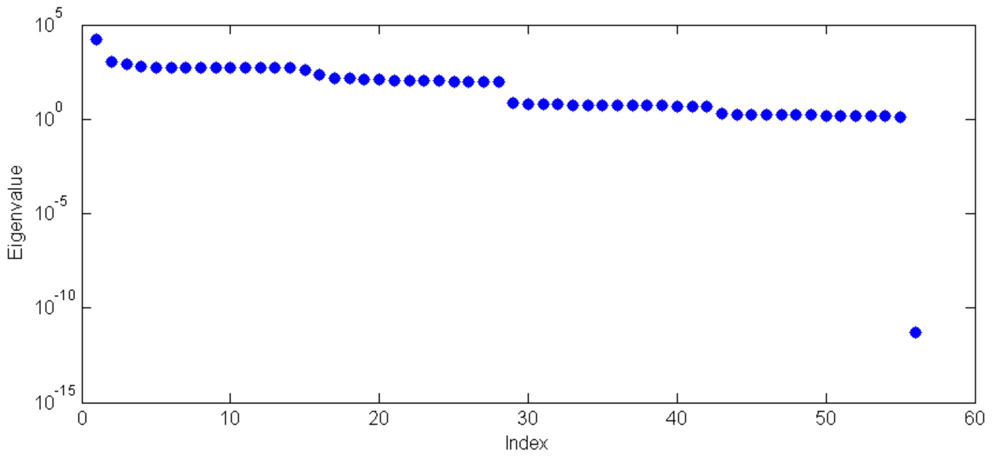

Figure 3.13: Plot of eigenvalues of fuselage sensor clippings

The eigenvalues returned by the SVD in the PCA computation in the discriminated and undiscriminated case are presented in Figures , , and . We do not observe any clear dropoff in the plot corresponding to the undiscriminated case for many indices, but we do see clearer jumps sooner in the discriminated cases.

3.4.3 Definition of Performance Metric

To evaluate the quality of the reconstructions given by each strategy, we first initialize the compression strategies, then establish $\Delta \subset \{1, \ldots, N\}$, a set of starting indices for testing with $|\Delta| = k$. We use care when choosing both our testing indices and our training indices, in order to ensure that all testing indices Δ are strictly greater than any indices in the training index set Γ, so as not to include any clippings in both the training and testing sets.

We then sort the columns of S according to our discrimination strategy $\Lambda_1, \Lambda_2, \Lambda_3$ and use Δ to draw a number of testing clippings from S_{Λ_i} in the same way as we did for our training clippings. We label these testing clippings $a_{(i,j)}$ for the ith sensor and jth clipping of that sensor. Each clipping a has dimensions 8192×1.

For each a, we compute $Q[a]$, for the SNR-based reconstruction quality function we will define below. For both the undiscriminated and discriminated case, we compute the average SNR-based quality as

$$\frac{1}{8k} \sum_{i=1}^{8} \sum_{j=1}^{k} Q[a_{(i,j)}]$$

Note that we exclude every temperature sensor from this error evaluation, as the constant nature of the temperature makes the signal uninterestingly simple to reconstruct. In our experiments with unsorted compression, we observed that the value of $Q[t]$ for some temperature clipping t was, reliably, much higher than that of the quality value for a non-temperature clipping. As a matter of fact, in the discriminated case, temperature clippings were reconstructed well enough to yield infinite values for $Q[t]$. As this invalidated any attempt to calculate average Q, we have not included temperature sensors in our quantitative evaluation of reconstruction quality.

We define the quality function $Q[a]$ as a mean-based calculation of signal to noise power ratio. In this case, the 'noise' is the error introduced by reconstruction in spectral magnitude of clippings. In terms of the FFTs of a and a^*, respectively $\hat{a}$

and $\hat{a}^*$:

$$
\begin{aligned}
Q[a] &= 10 \log_{10} \left(\frac{P_o}{P_e} \right), \text{ with} && (3.8) \\
P_o &= \frac{1}{8192} \sum_{i=1}^{8192} |\hat{a}_i|^2 \text{ and} && (3.9) \\
P_e &= \frac{1}{8192} \sum_{i=1}^{8192} |\hat{a}_i - \hat{a}_i^*|^2 && (3.10)
\end{aligned}
$$

3.4.4 Performance

To perform a thorough test of the performance of the three-classes discriminated compression algorithm as compared with the undiscriminated case or the entirely separated sensor case, a number of experiments were run. Five different datasets were generated and after each compression strategy was initialized, 100 test clippings were drawn, with care used to avoid using any clipping as both training and testing data.

Various numbers were chosen as the total basis size constraint, and the compression ratio was computed based on these. For each pair of one compression ratio and one compression strategy, the Q function was averaged over every clipping, sensor, and dataset. This yielded one single average SNR for every compression ratio and compression strategy, and the results were plotted in Figure 3.14.

The highlighted points of Figure 3.14 show that the discriminator raises the compression ratio from 45.52 to 58.53 with no loss in quality. The individual sensor compression strategy performs increasingly well at higher compression ratios, and as can be seen in the larger-scale plot in Figure 3.15, it eventually outperforms both other methods. The non-discriminated case is most prone to decline in quality, and

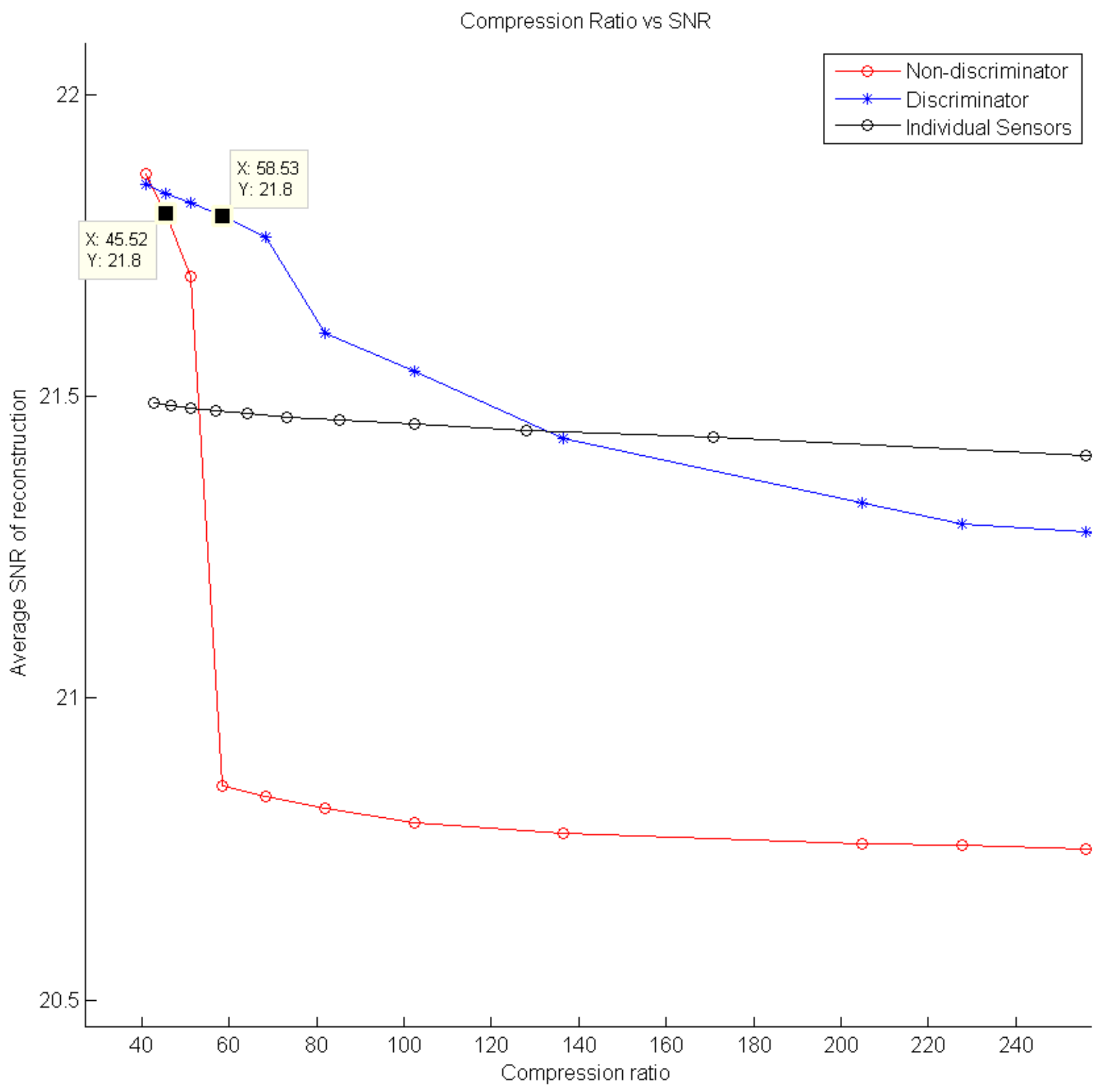

Figure 3.14: Plot of Compression Ratio vs SNRs

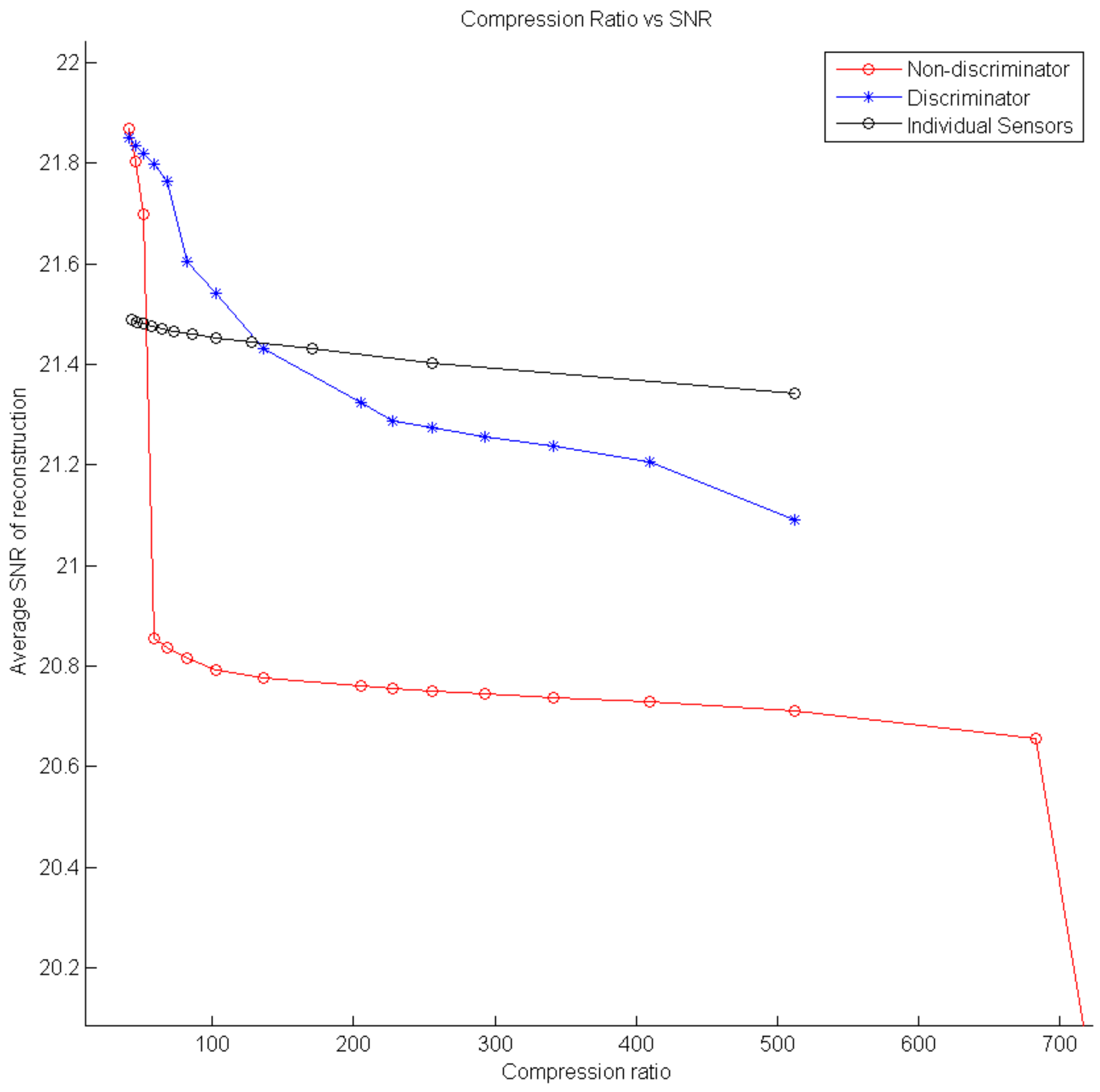

Figure 3.15: Large-scale Compression Ratio vs SNRs

while it starts out as good as the discriminated case and better than the individually compressed sensors, it quickly drops into last place.

Runtime tends to be greater in the discriminated case and greater still for individually compressed sensors, but the structure of the code is highly parallelizable. An extension of this work employing parallel computing would be highly informative; we plan to try this ourselves.

Chapter 4

Conclusion

4.1 Summary

In this paper, we set forth a discriminator-based modification to a compression algorithm. The compression algorithm is intended for use with sensor feeds for a jet health monitoring system. We implemented the algorithm and compared it to two other methods, by designing and creating a simulation of sensor feeds and evaluating the reconstruction performance of each grouping strategy on the simulated data.

Ultimately, we saw that our novel discriminator-based strategy does perform better than either other technique, in certain circumstances. It would appear that our algorithm may indeed be of true practical use in certain situations, though it is slower than one of the other methods tried.

4.2 Further Work

We expect that the runtime of our algorithm would be improved by parallelized computing; this has been supported by some preliminary testing. We intend to continue our experiment by adding parallelization support, as well as an automated discriminator. We would also like to see how the discriminator-aided compression works together with a fault-detection algorithm, and whether the discriminator is benificial or detrimental to its success. It would be most interesting to the author if the experiment could be repeated on real-world recorded data from a true jet flight. This would provide a much more compelling study.

One concern associated with the algorithm is the fact that it is most effective at compressing data clippings that are similar to those that arose previously and were used as training clippings to initialize the algorithm. If an entirely novel signal arises—perhaps in the case of a sudden, unprecedented and catastrophic mechanical failure—it may be too unlike the various previous training clippings to be captured well. If it is near orthogonal to our training basis, a dramatically different signal may go undetected.

Other possibilities for improvement or diversification of the experiment could involve: adding delays between the nodes of the vibration-spreading weighted graph, using dimensionality reduction tools other than PCA, not removing the compression means at all, substituting different types of noise for the white and pink noise used in this experiment, and adding additional wind-based noise sources to more points on the graph. It would also be highly informative to compute two identical datasets and then add the white noise only to one of them, and then see how well reconstructions from the noisy dataset match the noiseless dataset.

As a final point, we would like to repeat the experiment with a new type of discriminator classes; we suspect that we may see better results by grouping single branches of the weighted graph into compression classes. To explain this in terms of the weighted graph diagram Figure 3.2 from 3.2.4, we would use the classes (E1,E2), (F1,F2), (E3,E4), and (F3,F4). It is our belief that since the center node merges all vibrational data from all signals, the contribution to any one branch by way of this central conduit will be close to 1-dimensional. We would like to explore this idea.

Appendix A

Discussion of Noise Types

Herein we will define and discuss the discrete versions of the signals known as white and pink noise, the two types of noises used in our implementation. We will also compare them to brown noise, which is one name given to noise associated with Brownian motion.

Brown noise and white noise are well-studied, and form convenient reference points from which to describe pink noise. As we will see, pink noise forms a middle ground between the other two types of noise. The three types of noise are related by their spectra; each has magnitude $S(f) = k/f^p$ for frequency f and constants $k \in \mathbb{R}^+$ and $p \in \mathbb{Z}$. [6]

A.1 White noise

White noise characterized by *independent Gaussian-drawn samples.* It is a signal made up of a number of samples drawn identically and independently from a gaussian distribution. To express this in an equation, with $\bar{w}$ defined as a vector of n samples of white noise,

$$\forall i \in \{1, \dots, n\}, \quad \bar{w}_i \sim N(\mu, \sigma^2)$$

for some appropriately chosen μ and σ. The FFT of white noise is characterized by the exponent $p = 0$; thus, its FFT forms a uniform distribution $S(f) = k$. [6]

A.2 Brown noise/Brownian Motion

[6] Brown noise is characterized by independent Gaussian-drawn *increments.* It is the cumulative sum (or integral) of white noise; that is, the differences between successive samples are drawn iid from a gaussian distribution. Thus the vectorized form $\bar{b}$ of n samples has the following property:

$$\forall i \in \{1, \ldots, n\}, \quad \bar{b}_i - \bar{b}_{(i-1)} \sim N(\mu, \sigma^2)$$

again for some appropriately chosen μ and σ. Brown noise has a distrubition in the frequency domain characterized by the power exponent $p = 2$. This noise has magnitude of $S(f) = k/f^2$, where $k \in \mathbb{R}^+$ and f is the frequency at any given point, as previously discussed. [6]

A.3 Pink Noise

Pink noise, as stated, forms a middle ground between brown noise and white noise. It does not have a simple explanation in the time domain, but its spectra is easy to explain in terms of what we have already said. The power exponent p of pink noise is 1; thus, for pink noise, $S(f) = k/f$. [6] We take advantage of this fact in the paper by implementing pink noise by first defining the magnitude at every point of the FFT of our pink noise, and then adding randomized values for phase.

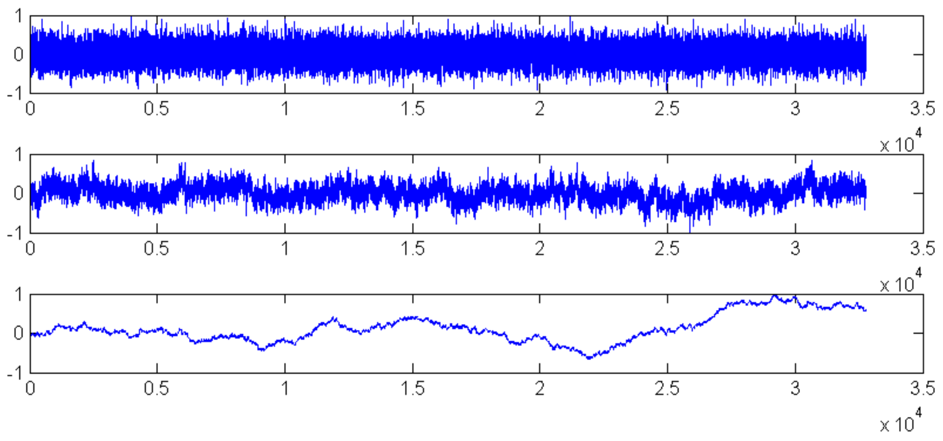

Figure A.1: Comparison of White, Pink, and Brown noise in time domain

A.4 Comparison of Plots

Experimental noise data was generated, and a series of plots of white, pink, and brown noise are provided in Figures A.1, A.2, and A.3. In each plot, white noise is pictured in the top subplot, pink noise in the center, and brown noise on the bottom. We note that linear interpolation does not imply meaning between mesh points as these signals are only defined on a discrete index set. A line plot was chosen solely because it shows clearer trends than a scatter plot.

The time-series data pictured in Figure A.1 demonstrates the behavior of each type of noise. White noise is entirely incoherent, which is consistent with its definition as independent samples. On the other hand, the value of each sample of brown noise is determined primarily by the immediately preceeding value, with only a small change introduced by each increment. As stated before, pink noise provides a middle ground between white and brown, and the time-series plot of pink noise supports this.

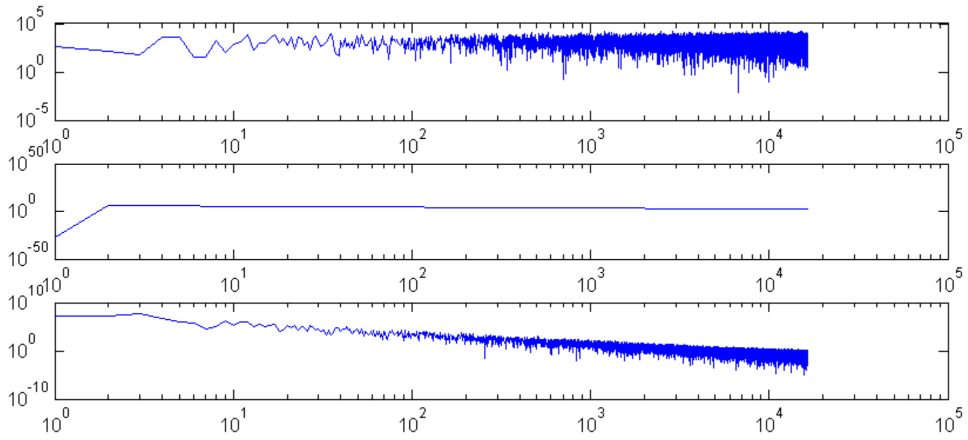

Figure A.2: Comparison of power spectrum of White, Pink, and Brown noise

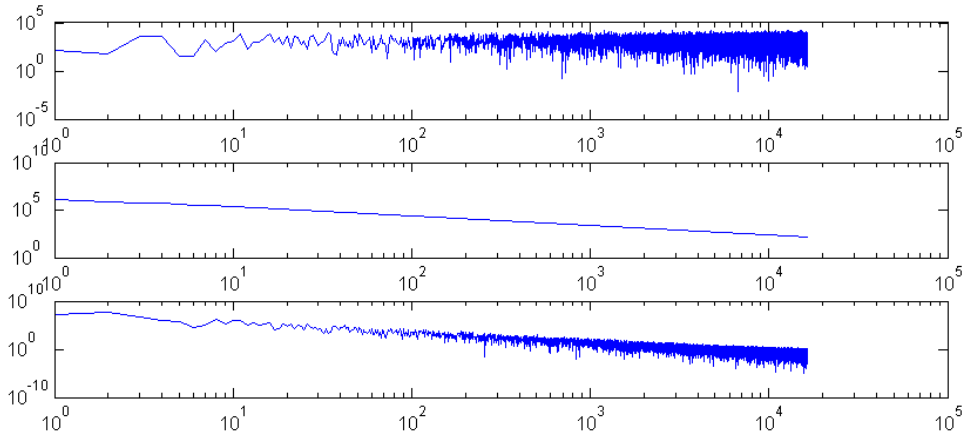

Figure A.3: Comparison of power spectrum, with first coefficient truncated

The power spectrum of each type of noise is pictured in Figures A.2 and A.3. The plots are logarithmic in both variables, which allows linear trends to appear. Once the first coefficient, corresponding to the constant component of each signal, is truncated, we are able to see that each magnitude spectrum takes the form of a noisy line of some slope, when viewed on a log/log plot. This provides another indication of the relationship between the three types of noise.

Appendix B

Source Code

We implement and run our experiment entirely in unmodified MATLAB R2010a. The following source code files encompass the whole of the code used in the production of this paper.

B.1 Data Generation

The following files generate the simulated data that we use to test our algorithm. 'A_MAIN_normal_plane.m' is the top-level script, which invokes each of the others.

B.1.1 A_MAIN_normal_plane.m

```
%% normal_plane.m :  Make database

%{
This file generates a single ordinary plane, with 4 engines that
include noise and with fuselage noise.  There are 4 engine sensors
and 4 fuselage sensors, and 4 temperature sensors.

The temperature sensors are a self-contained system, but the
fuselage and engine sensors hear some of the same vibrational data.

The engine sound comes through strong on the engine sensors and
weakly on the fuselage sensors, and vice versa for the fuselage
sound.

%}

%% init
tag = 'normal_plane';
filename = tag;
basic_plane

%% tweak

%Make any desired adjustments to parameters here, i.e. introduce
%volume increases indicating fault conditions

%% sense/mix
mix_sensors

%% save
%Uncomment to save a .mat file for later use
%save(filename);
beep
'Database complete'
```

B.1.2 basic_plane.m

```
%% basic_plane.m
%{

use the engine_sound function four times with same values except for slight speed va
4 engines
4 wing sensors,
4 fusel. sensors
4 temp. sensors

multiply the engine matrix by the mixing matrix

Engine sounds matrix has a row for each engine and a column for each tick.
Mixing matrix has a row for each sensor and a column for each engine.

Mixing * Engine = [sensor x engine][engine x tick] = [sensor x tick]



Highest freq is going to be inner_rpm * inner_blades / 60 sec/min,
which is 3.3786e+003 or about 4000.
Therefore Nyquist freq. is less than 8000 samp/sec.  We'll use
8192 = 2^13 because it's a power of two and plays nice with FFTs.

We add a little bit of randomness to the loudness
of gears, turbine speed, etc.
%}


%% global init
samp_rate = 2^13;
rng = gmdistribution(0,1,1);
ticks = ((1:2^19)-1)';    %our set of times

%% randomization init
%these parameters will take the form of a constant describing the maximum
%multaplicative difference between a specified parameter and the randomized
%one; for example, if 50 is specified and .1 is the randomization param,
%then the actual value will range from 50*.9 = 45 to 50*1.1 = 55.
%The multiplicative randomization factor will be distributed uniformly.

%When creating the multiplicative factor for scaling the specified param,
%do it in the form 1+(rand_param*2*rand - rand_param).

gear_rand = .05;
out_rpm_rand = .05;
in_rpm_rand = .05;
eng_noise_rand = .05;
fus_noise_rand = .05;

%% Engine parameters init
gear_vol = 10^(-3/20);
```

```
gear_properties = [1 2 2; 5/13 7/4 1/2; ...
    gear_vol * (1+ gear_rand * 2 * rand -gear_rand) ...
    gear_vol * (1+ gear_rand * 2 * rand -gear_rand) ...
    gear_vol * (1+ gear_rand * 2 * rand -gear_rand)];

%see noise section for noise volume

base_temp = 696; %in celsius

outer_rpms = 2069;
inner_rpms = 9653;
outer_vol = 1;
inner_vol = 1;
outer_blades = 20;
inner_blades = 21;
out_blade_vol = 1;
in_blade_vol = 1;

%{
        FORMAT:
OUT RPMS        IN RPMS
OUT VOLUME      IN VOLUME
OUT BLADES      IN BLADES
OUT BLADE VOL   IN BLADE VOL
%}

turbine_A_properties = [outer_rpms*(.95+.1*rand) ...
```

```
    inner_rpms*(.95+.1*rand); outer_vol inner_vol; ...
    outer_blades inner_blades; out_blade_vol in_blade_vol];
turbine_B_properties = [outer_rpms*(.95+.1*rand) ...
    inner_rpms*(.95+.1*rand); outer_vol inner_vol; ...
    outer_blades inner_blades; out_blade_vol in_blade_vol];
turbine_C_properties = [outer_rpms*(.95+.1*rand) ...
    inner_rpms*(.95+.1*rand); outer_vol inner_vol; ...
    outer_blades inner_blades; out_blade_vol in_blade_vol];
turbine_D_properties = [outer_rpms*(.95+.1*rand) ...
    inner_rpms*(.95+.1*rand); outer_vol inner_vol; ...
    outer_blades inner_blades; out_blade_vol in_blade_vol];
engine_temps = ones(1,4) * base_temp;

%indices here are inner/outer, parameter, engine#
turbine_properties = cat(3, turbine_A_properties, turbine_B_properties, ...
    turbine_C_properties, turbine_D_properties);

%%  Noise init

%originally just 1; volume of the pink fuselage noise
fuselage_volume = 1e3;

%volume of the white noise at each engine
noise_volume = .3;


%% after this script:
```

```
%continue by introducing any desired faults,
%and then by mixing sensor data
```

B.1.3 engine_sound.m

```
%% engine_sound.m
%{
This file was converted from a script to a function.  It was
originally designed to create a noise that sounds like a jet.  It is now a
function that can be called to do the same thing, but has many parameters
to control exactly how that will happen.

param.s: matrix of gear linkages/ratios/noise levels

Number of engines needn't be given as this function can be called multiple
times, and also it will be implicit in the dimension of the square mixing
matrix when we start using multiple engines.

Turbine_properties is a matrix of column vectors. each vector contains,
in order, turbine rpms, turbine volume, number of blades, blade volume.
If turbine_properties is m by n, there are n turbines and only the first
four rows are used.

Gear_properties is also a matrix of column vectors. each vector contains,
in order, gear linkage index, gear ratio, gear volume.
%}
```

```
function foo = engine_sound(ticks, rng, turbine_properties, ...
    gear_properties, noise_volume)
%% Init everything
gear_sounds = 0;        %this will remain 0 if no gears are specified

%% Establish properties of the turbines

turbines = size(turbine_properties,2);
freqs = turbine_properties(1,:) / 60;   %convert from RPMs to Hz
blades = turbine_properties(2,:);

%% Generate sounds for each turbine

sine_turbines = zeros(size(ticks));
sine_blades = sine_turbines;

for k = 1:turbines
    temp_freq = 2*pi*freqs(1,k);
    shaft_vol = turbine_properties(2,k);
    blade_vol = turbine_properties(4,k);
    sine_turbines = sine_turbines + sin(ticks * temp_freq) * shaft_vol;
    sine_blades = sine_blades + sin(ticks * temp_freq * blades(k))*blade_vol;
end


%% Generate gear sounds
```

```
if exist('gear_properties', 'var')
    connections = gear_properties(1,:); %which turbine is the gear connected to?
    gear_ratio = gear_properties(2,:);  %what's the gear differential?
    gears = length(connections);    %number of gears
    gear_sounds = zeros(length(ticks), gears);
    gear_freq = gear_sounds;
    for k = 1:gears
        gear_freq(k) = freqs(connections(k)) * gear_ratio(k);
        temp_vol = gear_properties(3,k);
        gear_sounds(:, k) = sin(2*pi*gear_freq(k)*ticks) * temp_vol;
    end
    gear_sounds = sum(gear_sounds, 2);
end

%% Generate noise

%establish some noise according to specified distribution
noise = random(rng,length(ticks));

noise = noise * noise_volume;




%% combine all the signals
foo = sine_turbines + sine_blades + gear_sounds + noise;
```

B.1.4 fuselage_sound.m

```
%% fuselage_sound.m
%{
This file was adapted from jet_sound.m, which generates engine noise, to
be a function that generates fuselage wind noise.

It takes in some basic params about the environment we're using as well as
the parameters for noise, consisting of volume and frequency envelope.
%}
function fus = fuselage_sound(ticks, noise_volume)

%% establish a 1/f power spectrum
pow = 1./(ticks(1:end/2)+1);
mag = sqrt(pow);
flip_mag = mag(end-1:-1:1);
mag = [0; mag; flip_mag];



%% establish randomized phase
phase = exp(2*pi*1i*rand(length(ticks)/2, 1));
flip_imag = phase(end-1:-1:1);
flip_imag = real(flip_imag) - 1i*imag(flip_imag);
phase = [1; phase; flip_imag];

%% package and return it
fus_transf = mag.*phase;
fus = noise_volume * real(ifft(fus_transf));
```

B.1.5 mix_sensors.m

```
%% mix_sensors
%{
This file picks up where the param init scripts leave off.
It takes the various vars that have been init'ed and uses them
to generate the data of the entire plane's sensors.
%}


%% establish mixing procedure
%reduction from one engine to the next
a = 10^(-10/20);

%reduction across the fuselage
b = 10^(-16/20);

%reduction toward nose of fuselage
c = 10^(-18/20);

%reduction toward tail of fuselage
d = 10^(-13/20);


%{
one sensor on each engine, four on fuselage.
order is: left outer, left inner, right inner, right outer,
nose, front mid, back mid, tail.
Columns are first thru fourth engine and then fuselage noise.
%}
mixing = [...
    1 a a*b*b a*b*b*a; ...
    a 1 b*b b*b*a; ...
    a*b*b b*b 1 a; ...
    a*b*a b*a a 1; ...
    a*b*c^2 b*c^2 b*c^2 c^2*b*a; ...
    a*b*c b*c b*c c*b*a; ...
    a*b*d b*d b*d d*b*a; ...
    a*b*d^2 b*d^2 b*d^2 d^2*b*a ...
    ];

fuselage_noise_mixer = [a*b*c^2 b*c^2 c^2*b c^2*b*a 1 c c*d c*d*d]';

mixing = [mixing fuselage_noise_mixer];

%% generate raw unsensed engine data
num_engines = size(turbine_properties, 3);
engine_data = zeros(length(ticks), num_engines);
for k = 1:num_engines

    foo = engine_sound(ticks, rng, turbine_properties(:,:,k), ...
        gear_properties, noise_volume);
    engine_data(:,k) = foo;
end


%% generate fuselage data
fuselage_data = fuselage_sound(ticks, fuselage_volume);


%% generate temperature data
temperature_data = repmat(engine_temps, size(engine_data,1), 1);



%% Mix the generated signals
%the temperature data does not have any kind of mixing applied to it,
%so we just append it after we do the mixing
premix_data = [engine_data fuselage_data];
mixed_data = premix_data * mixing';

sensor_data = [mixed_data temperature_data];
```

B.2 Algorithm

These source files were used for analyzing the data. The top-level function 'large_comparison.m' calls each of the others. For our evaluation of the algorithm's performance, we used this file with a varying range of parameters, and slight modifications.

B.2.1 large_comparison.m

```
%% large_comparison.m
%{
This file generates 30 different datasets and runs 10 trials on each of
them.  It checks to see which method (grouped/ungrouped) does best SNR-wise
on the compression algorithm.
%}



basis_sizes = 100:-10:20;
sub_trials = 10;

%take note of how the compression is actually doing
clipping_size = 2^13;
num_coeffs = clipping_size./2 + 1;
compression_rate = num_coeffs./basis_sizes;
```

```
avg_unsorted = zeros(size(basis_sizes));
avg_sorted = avg_unsorted;

exec_time_unsorted = avg_unsorted;
exec_time_sorted   = avg_unsorted;



sub_avg_ungrouped = zeros(1,3);
sub_avg_grouped = sub_avg_ungrouped;
sub_exec_time_unsorted = sub_avg_ungrouped;
sub_exec_time_sorted   = sub_avg_ungrouped;

for iter = 1:length(basis_sizes)
    basis_size = basis_sizes(iter)
    fus_basis_size = basis_size/2;
    eng_basis_size = fus_basis_size-1;
    %tmp_basis_size = 1;

    for sup_trial = 1:3

        run data_generation_scripts\A_MAIN_normal_plane.m

        tic
        snrmat_all = compression_test(sensor_data, basis_size, sub_trials);
        sub_exec_time_unsorted(sup_trial) = toc;

        %EXCLUDE the temp sensors from the averaging
        sub_avg_ungrouped(sup_trial) = mean(mean(snrmat_all(:,1:8)));


        tic

        snrmat_eng = compression_test(sensor_data(:,1:4), eng_basis_size, sub_trials

        snrmat_fus = compression_test(sensor_data(:,5:8), fus_basis_size, sub_trials

        sub_exec_time_sorted(sup_trial) = toc;

        %%%snrmat_tmp = compression_test(sensor_data(:,9:12), tmp_basis_size, sub_tr


        snrmat_agg = [snrmat_eng snrmat_fus];

        sub_avg_grouped(sup_trial) = mean(mean(snrmat_agg));


    end


    avg_unsorted(iter) = mean(sub_avg_ungrouped);

    avg_sorted(iter) = mean(sub_avg_grouped);


    exec_time_unsorted(mean(sub_exec_time_unsorted));

    exec_time_sorted(mean(sub_exec_time_sorted));

end


beep

beep

beep

'DONE!!'

%uncomment to save .mat of results

%save('large_comparison_results')
```

```
return

%% display output
%invoke for standard plot

figure
title('SNR vs Compression Ratio')
hold on
plot(avg_unsorted, compression_rate, 'r—o')
plot(avg_sorted, compression_rate, 'b—*')
xlabel('Average SNR of reconstruction')
ylabel('Compression ratio')
legend('Unsorted compression', 'Sorted compression')

figure
title('Compression Ratio vs Execution Time')
hold on
plot(compression_rate, exec_time_unsorted, 'r—o')
plot(compression_rate, exec_time_sorted, 'b—*')
xlabel('Compression ratio')
ylabel('Execution time')
legend('Unsorted compression', 'Sorted compression')
```

B.2.2 compression_test.m

```
%% Compression_test.m
```

```
%{
Provide this function with recordings from at least one sensor,
and specify other parameters as well, to test how well the algorithm
does under those conditions.

%}

function snrmat = compression_test(sensor_data, basis_size, tests)



%% init

%This code requires that the input matrix position the individual sensors
%vertically.
[samples sensors] = size(sensor_data);
clip_length = 8192;

training_clippings = ceil(basis_size / sensors) * 2;
non_training_start = training_clippings + 1;
num_clippings = tests + training_clippings;

%% establish clipping spacing

%this establishes uniform spacing between clippings.
interval = samples/(num_clippings + 2);

%make a list of places to start.
```

```
%this may lead to problems if clipping length is very long or
%num_clippings is too large, causing us to run past the end of
%sensor_data in forming clippings
start_list = round((0:num_clippings-1)*interval + 1);

%% store clippings (NOT their fft's)
clippings = cell(1, num_clippings);
for k = 1:num_clippings
    start = start_list(k);
    %make a list of clippings
    clippings{k} = sensor_data(start:start+clip_length-1, :);
end

%% %%%%%%%%%%%%  establish compression basis

%% gather judge clippings


judge_clippings = zeros(clip_length, sensors * training_clippings);
for k = 1:training_clippings
    judge_clippings(:, 1+(k-1)*sensors:k*sensors) = clippings{k};
end
judge_fft = abs(fft(judge_clippings));

%% remove means (results in a column vector)
compression_mean = mean(judge_fft, 2);
demeaned = judge_fft - repmat(compression_mean, 1, sensors * training_clippings);
[pre_basis eigvals] = custom_pca(demeaned, basis_size);
```

```
compression_basis = pre_basis;



%% normalize compression basis
%Depending on the implementation of PCA, this may already have happened
for k = 1:basis_size
    temp = compression_basis(:,k);
    temp_pow = sum(abs(temp).^2);
    compression_basis(:,k) = compression_basis(:,k) / sqrt(temp_pow);
end

%% Compress AND decompress, and compute SNR

compressed_clippings = cell(1,tests);
reconstructed_ffts = cell(1,tests);

%average SNR for each sensor during each trial
snrmat = zeros(tests,sensors);

%iterate across every trial clipping
for k = 1:tests


    %make a list of clippings; make each col be a clipping
    %get a list of the top 28 eigenvectors for one of the
    %clipping's FFT's SVD.  make this into a matrix of columns,
    %each column being an eigenvector.
```

```
%multiply the transpose of the clipping column matrix by the
%eigenvector matrix to get a matrix whose rows comprise the
%eigendecomposition coefficients of each of the many sensor rows.

%SKIP all training clippings by adding an offset here
orig = clippings{k + training_clippings};
orig_fft = abs(fft(orig));

%REMOVE MEAN
orig_demeaned = orig_fft - repmat(compression_mean, 1, sensors);

%compute coefficients
compressed_clipping = compression_basis' * orig_demeaned;
compressed_clippings{k} = compressed_clipping;

%compute reconstruction
reconstructed_fft = compression_basis * compressed_clippings{k};

%REPLACE MEAN
reconstructed_fft = reconstructed_fft + repmat(compression_mean, 1, sensors);

reconstructed_ffts{k} = reconstructed_fft;

%get average SNR for this clipping
temp = 0;
```

```
    for sensor = 1:sensors
        %NOTE that we must compute the SNR based on the abs. of the
        %fft, or else we may gain error based on any phase
        %distortion, which does not matter to us.
        foo = sig_to_noise(abs(orig_fft(:,sensor)), ...
            abs(reconstructed_fft(:, sensor)));

        snrmat(k,sensor) = foo;
        temp = temp + foo;
    end

end



end
```

B.2.3 custom_pca.m

```


%{
This function will NOT subtract the mean for you; you have do to that
manually before calling this, if you want it done.
%}

function [eigenvectors D] = custom_pca(sensors, dims)

[U D V] = svd(sensors);
```

```
clearvars V;

[a b] = size(D);

if a ~= 1 && b ~= 1

    D = diag(D);

end

eigenvectors = U(:,1:dims);




end
```

B.2.4 sig_to_noise.m

```
%{

The higher this is, the better we did.

signal divided by noise:

log10 of the square of the ratio of avg signal amp. to avg noise amp.

power is mean of squares of signals

%}

function ratio = sig_to_noise(orig, altered)




if any(size(orig) ~= size(altered))

    size(orig)

    size(altered)

    error('Improper sizing!')

end
```

```
stop = size(orig,2);
ratio = zeros(1,stop);

for k = 1:stop
    %get what we'll compare to
    base = orig(:,k);
    %define error
    err = base - altered(:,k);

    %define power
    pre_Pe = abs(err).^2;
    pre_Po = abs(base).^2;
    Pe = mean(pre_Pe);
    Po = mean(pre_Po);

    ratio(k) = 10 * log(Po/Pe)/log(10);
end
```

Bibliography

[1] A. Abbasion, A. Rafsanjani, A. Farshidianfar, N. Irani. "Rolling element bearings multi-fault classification based on the wavelet denoising and support vector machine." Mechanical Systems and Signal Processing. (March 2007)

[2] Massimo Cavacece and Alberto Introini. "Analysis of Damage of Ball Bearings of Aeronautical Transmissions by Auto-Power Spectrum and Cross-Power Spectrum." J. Vib. Acoust. Vol 124. 180 (2002)

[3] Thomas M. Cover and Joy A. Thomas. Elements of Information Theory. Hoboken, NJ: Wiley, 2006.

[4] Michael R Dellomo. "Fault detection in helicopter gearboxes using neural networks", J Acoust. Soc. Am. Vol. 91. 4 (Apr 1992) 2359.

[5] Lisa A Griffin. "Comparison of the speech transmission index and the modified rhyme test in simulated cockpit ambient noise." J Acoust. Soc. Am. Vol. 91. 4 (Apr 1992) 2328.

[6] John M. Halley. "Ecology, evolution, and $1/f$-noise." TREE Vol. 11. 1 (January 1996) 33–37

[7] Paul Hayton, Simukai Utete, Dennis King, Steve King, Paul Anuzis, Lionel Tarassenko. "Static and Dynamic Novelty Detection Methods for Jet Engine Health Monitoring." Philosophical Transactions: Mathematical, Physical and Engineering Sciences Vol. 365. 1851 (15 Feb 2007) 493–514

[8] C. E. Shannon. "Communication in the presence of noise", Proc. Institute of Radio Engineers. Vol. 37, no. 1, (Jan. 1949) 10–21

[9] John A. Lee and Michel Veryelsen. Nonlinear Dimensionality Reduction. New York: Springer, 2007.

[10] Mladen Victor Wickerhauser. Mathematics for Multimedia. Boston: Birkhauser, 2004.

[11] Zengbing Xu, Jianping Xuan, Tielin Shi, Bo Wu, Youmin Hu. "Application of a modified fuzzy ARTMAP with feature-weight learning for the fault diagnosis of bearing." Expert Systems With Applications. Vol. 36, no. 6. (2009) 9961–9968

[12] Joe Yoon. "Jet Engine Types." Aerospaceweb.org. (1 July 2001) Accessed 22 Apr, 2012 <http://www.aerospaceweb.org/question/propulsion/q0033.shtml>

[13] GE Aviation. "Engines 101." Accessed Oct 2011 <http://www.geaviation.com/education/engines101>.