SMJ-A09B

Bird Call Identifier

Identifying Songs of Bird Species through Digital Signal Processing Techniques

A Major Qualifying Project submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the degree of Bachelor of Science

Submitted by:

Tyler Carroll
Rose Colangelo
Tom Strott

Submitted to:

Project Advisor:
Professor Susan Jarvis

29 Apr 2010

This report represents work of WPI undergraduate students submitted to the faculty as evidence of a degree requirement. WPI routinely publishes these reports on its web site without editorial or peer review. For more information about the projects program at WPI, see http://www.wpi.edu/Academics/Projects.

Abstract

The purpose of this Major Qualifying Project is to create a device that identifies bird calls in the wild. The aim of this project is to create a handheld device that will be able to interpret bird calls using a high quality microphone and various signal processing techniques and display the top matches on an LCD screen to the user. The main objective of this project is to be able to identify the songs of several bird species in central Massachusetts on a lab development board while additional work would include downloadable software that would add the bird songs of more bird species and the conversion of from a lab development board to a handheld device.

Authorship

The three team members contributed equally to the work on this project and to the report.

Acknowledgements

The team would like to thanks the following people for their contributions to the project:

Susan Jarvis, Adjunct Instructor, WPI

Mike Webster, Director of the Macaulay Library

Michael Young, Sound Technician, Macaulay Library

Christine Drew, Manager, Instruction & Outreach, Gordon Library

Table of Contents

Abstract....... 2

Authorship 3

Acknowledgements 4

Table of Figures 7

Table of Tables 8

Table of Equations 8

Chapter 1: Introduction 9

Chapter 2: Background 12

2. 1 Prior Art 12

2.2 Signal Processing 14

2. 3 Mel-frequency Cepstrum 16

2.4 Bird Call Recognition 18

Chapter 3: Methodology 20

3.1 Scope 20

3.2 System Block Diagram 21

3.3 Creation of an Algorithm 23

3.4 Obtaining Samples 24

3.5 Modular Design Choices 26

3.5.1 Netbook Processor 27

3.5.2 Front End Filter 28

3.6.3 Digital Signal Processing Chip 28

3.6.4 LCD Screen 29

3.6.5 Microphone 30

3.7 Algorithm Implementation 30

3.7.1 Filtering and Frame Generation 31

3.7.2 Windowing Concepts 32

3.7.3 Mel-Scale Filtering 33

3.7.4 Correlation and Database Implementation 36

Chapter 4: Results 38

4.1 MATLAB Testing Results 38

4.2 Testing the Algorithm in C 43

4.2.1 Fast Fourier Transform Test.. 43
4.2.2 Discrete Cosine Transform Test 45
4.2.3 Correlation Function 47
4.2.4 Hamming Window Function 50
4.2.5 Testing the MFCC Algorithm 52
4.3 Obstacles 54
4.3.1 Record and Playback Function 54
4.3.2 Front End Filtering 55
4.3.3 Complex Numbers 56
4.3.4 Fast Fourier Transform 57
4.3.5 Memory Problems... 57
4.3.6 Noise... 59
Chapter 5: Conclusions 61
5.1 Discussion of MATLAB Results 61
5.2 Discussion of C Results 61
5.3 Future Work Recommendations 62
5.3.1 MATLAB Future Work 62
5.3.2 C Implementation Future Work 62
5.3.3 Hardware Design Future Work 63
Appendices 65
Appendix A: MATLAB Test Results 65
Appendix B: MATLAB Source Code 85
Bird Finder with Database 85
MFCC Comparison Code.. 85
Appendix C: DSP Source Code 87
source.c 87
kannumfcc1.2.c 98
fft.c 102
corr.c 102
mfcc_bank.c 105
References 115

Table of Figures

Figure 1: System Block Diagrams 21
Figure 2: Cape May Warbler Song Comparison With Background Noise in Signal 1 26
Figure 3: Magnitude Response of Lowpass Filter..... 28
Figure 4: Block Diagram of C6713 DSK 29
Figure 5: Algorithm Flow Chart 32
Figure 6: Frequency Domain Filtering Flow Chart..... 35
Figure 7: Comparison of Two Calls from the Same Carolina Wren 39
Figure 8: Comparison of Two Human Whistle Trills 40
Figure 9: Magnolia Warbler Comparison with One "Unknown" Song 41
Figure 10: Carolina Wren Comparison with One "Unknown" Song 42
Figure 11: 2 kHz Sine Wave..... 44
Figure 12: Fast Fourier Transform of 2 kHz Sine Wave 45
Figure 13: DCT of Input Samples in C 46
Figure 14: DCT of Input Samples in MATLAB 47
Figure 15: Original Input Signal 48
Figure 16: Time Shifted Input Signal..... 48
Figure 17: Rectangular Windowed Cross Correlation of Input Signals..... 49
Figure 18: Wrap Around Cross Correlation of Input Signals 50
Figure 19: Hamming Window Function in C 51
Figure 20: Hamming Window Function in MATLAB 52
Figure 21: Cape May Warbler song comparison 66
Figure 22: Cape May Warbler song comparison 67
Figure 23: Cape May Warbler song comparison 68
Figure 24: Cape May Warbler song comparison 69
Figure 25: Cape May Warbler song comparison 70
Figure 26: Magnolia Warbler song comparison..... 71
Figure 27: Magnolia Warbler song comparison..... 72
Figure 28: Magnolia Warbler song comparison..... 73
Figure 29: Magnolia Warbler song comparison..... 74
Figure 30: Magnolia Warbler song comparison..... 75
Figure 31: Mourning Warbler song comparison 76
Figure 32: Mourning Warbler song comparison 77
Figure 33: Mourning Warbler song comparison 78
Figure 34: Mourning Warbler song comparison 79
Figure 35: Carolina Wren song comparison..... 80
Figure 36: Carolina Wren song comparison..... 81
Figure 37: Carolina Wren song comparison..... 82
Figure 38: Carolina Wren song comparison..... 83
Figure 39: Carolina Wren song comparison..... 84

Table of Tables

Table 1: System Requirements for 32-Bit MATLAB 27
Table 2: Bird song average correlations and percentages choosing the correct bird 43
Table 3: Cape May Correlations 53
Table 4: Carolina Wren Correlations 53
Table 5: Magnolia Warbler Correlations 53
Table 6: Mourning Warbler Correlations 53

Table of Equations

Equation 1: Discrete Fourier Transform 15
Equation 2: Cross Correlation 16
Equation 3: Conversion from Hertz into Mel 17

Chapter 1: Introduction

Various species of birds have unique bird calls. These bird calls are distinct based on inflection, length, and context, meaning the same bird may have more than one call. A device that would analyze the signal and identify the bird based on the bird call could be of tremendous help to an ornithologist. This project proposed the development of this device using signal processing and embedded design. The first task was to find or create a database of high-quality bird calls to use for identification. Using this database, the team compared various features of the bird calls of a certain species and ascertained the features which distinguish that species from other species. Using these features, a recorded bird call was identifiable as a species of bird.

This project is an important effort because ornithology is not always an exact science in the field; it is based on the interpretation of the scientist hearing the bird's song. A device that could quantitatively match signal waveforms would make the science more exact. Furthermore, bird watching is a hobby that many people enjoy. The ability to identify birds could increase the enjoyment of bird watching enthusiasts everywhere.

Before discussing the project further, it is important to delve into the prior art related to the project to gain an understanding of how current products work and what needs improvement. Several products which identify bird songs already exist, but none of them currently in production do exactly what was attempted in this project. There is a wealth of information on the internet as well as some handheld devices that require the user to match the bird calls. A discontinued product exists that does digital signal processing of the bird call and displays likely species, and the team intends to produce something in that vein with some improvements.

In order to create a bird call identification device, the team needed to correctly utilize several signal processing techniques. Some of these techniques included filters, discrete Fourier

transforms, cross-correlation, wavelets, cepstral analysis, and audio spectrograms (Cai et al., 2007; Lee et al., 2006). Filters were necessary to improve the quality of the bird songs and remove any unwanted noise. Bird songs cover a wide frequency range and discrete Fourier transforms allowed the team to analyze the different frequencies in each call. Cross-correlation allowed the team to compare recorded bird calls with the bird call database, both in time and frequency. Next, the group used discrete wavelet transforms. An advantage that discrete wavelet transforms (DWT) have over Fourier transforms is temporal resolution; a DWT captures both frequency and time information. Additionally, the team created audio spectrograms using both Fourier and discrete wavelet transforms to examine each bird's song. Lastly, the group used mel-frequency cepstral coefficients. To find these mel-frequency cepstral coefficients, the Fourier transform is taken and then the power of the spectrum is mapped onto the mel scale (Lee et al., 2006). At that point, the discrete cosine transform (DCT) of the mel logarithms of the power spectrum is taken, and the resulting amplitudes are the mel-frequency cepstral coefficients. Each technique manipulated the bird call in a different way to help the group identify which species the call originated from.

One challenge that was faced during the attempts to identify actual bird songs was recognizing all calls made by a particular species of bird. A bird's song can contain a significant amount of information including the bird's species, sex, individual identity, his territorial and reproductive status, and his probability of responding aggressively or sexually to a potential recipient (Emlen, 1972). There is so much information that call variation is inevitable among birds and any one bird can have four or more songs in its repertoire. This was one of the group's greatest challenges because understanding how birds vary their songs was a significant roadblock to understanding how to differentiate species.

The previous attempts to create a device that could identify a bird's species by its call have been expensive or don't meet all the requirements of an ornithologist or bird watching enthusiast. This project attempted to improve upon previous devices' shortcomings. The goal of this project was to use signal processing and embedded systems to identify bird species by their calls and lay the foundation for a hand-held bird call identification device.

Chapter 2: Background

This chapter is intended to educate the reader about background information pertaining to bird vocalization and signal analysis in order to instill a greater understanding of the project. The prior art in the area of bird vocalization analysis is first discussed, followed by signal processing techniques, and concluded with bird call variations.

2. 1 Prior Art

Before discussing details of the project, it is important to first explore the prior art that is related to the goal. Many products exist on the marketplace, but none that are currently in production are quite like the planned project. There are two main reasons for this. First of all, signal processing is a complicated process, often requiring an expensive processing chip. From a production standpoint, it would be less expensive to manufacture a product without a signal processing chip. A popular way of doing this is recording the signal and having the user do the signal processing, comparing the known bird call with the unknown bird call and determining whether or not they match. This will be discussed in detail later on.

One product on the market is an Apple application called iBird Explorer ("Compare Birding Apps"), for use with an iPod Touch or an iPhone. This application is essentially a field guide for birds complete with bird calls. It shows pictures and matches them with bird calls, and also allows the user to search for birds based on a number of factors including color, shape, habitat, and location.

Other products on the market are standalone devices that do not have the same wealth of information, but still match a species of bird with a pre-recorded bird call from that bird. One such product is the IdentiFlyer, which is a handheld device that allows for headphone use. There are ten buttons up the sides with pictures next to them that allows the user to hear the bird call of

that particular bird. The device is expandable, allowing for more than ten bird species with additional purchases ("Identiflyer").

Another relevant source of information is eNature.com. This website contains a field guide for birds that includes species pictures, field descriptions, range maps, and bird calls. Bird searches are based on shape, color, size, region, and habitats. Although the information is useful, internet access is needed to access the information, and that may not be available in the field ("eNature").

The prior art that is most similar to the project is called the Song Sleuth. This device is made by Wildlife Acoustics and is a handheld device that implements a directional microphone to receive bird calls. The bird calls are then analyzed by a digital signal processing circuit board, and the top three most likely bird species from the database are listed in order. The database is taken from Cornell University's Macaulay Library. It lists as a $300 device, but production has been discontinued ("Song Sleuth").

Wildlife Acoustics also makes other products designed for bird song identification. The current model is called the Song Meter 2, or SM2. It is a weather resistant recording device that can record many hours of wildlife sound. Wildlife experts then take the memory to a computer and analyze the signals with proprietary software known as Song Scope. Song Scope automatically identifies various bird, frog, and other wildlife sounds while filtering out other noise or other wildlife sound that occurs simultaneously ("Song Meter 2").

This project was most closely related to the work of Wildlife Acoustics' work, although the implementation differed. This research into prior art has shown that similar projects have been done, but the one that is most like the team's idea is no longer in production. The Song

Sleuth was discontinued because of the greater functionality of the Song Scope computer software, but future implementations of the team's device may be able to implement various ideas into their software to make it more robust, such as narrowing down the possible database hits by requiring the user to input a region or coloration. Concepts like this allow the team to make a robust handheld platform for analyzing signals without needing the superior processing of a computer.

2.2 Signal Processing

In order to create a bird call identification device, the team needed to correctly utilize several signal processing techniques. Some of these techniques included filters, discrete Fourier transforms and cross-correlation. Each technique manipulated the bird call in a different way to help identify which species the call originated from. The filters were used to filter the incoming bird call and remove the noise that would disturb the results. Discrete Fourier transforms transformed the signals into the frequency domain so the team was able to examine the different frequencies within each signal. Cross-correlation let the group compare the incoming bird call with their database of bird calls, both in the time and frequency domains, and match the signals with the highest correlations to determine the most likely matches. The majority of the signal processing was developed first using MATLAB then eventually implemented in C on a digital signal processing chip.

The first signal processing technique necessary to identify a bird call is filtering. Filtering is necessary because it is likely that the bird call will contain a certain amount of noise. Additive noise could adversely affect the results. There are several types of filters that the team could have used to filter noise. Some basic filters are low-pass filters, high-pass filters, band-pass filters and band-stop filters. Low-pass filters function by passing low-frequency signals and attenuating or

reducing the amplitude of high-frequency signals that are higher than the filters designed cutoff frequency. High-pass filters function the opposite way by passing high-frequency signals and attenuating frequencies lower than the cutoff frequency. Band-pass filters pass frequencies within a certain range and reject all frequencies outside that range. These filters can be created by combining a low-pass filter with a high-pass filter. The opposite of a band-pass filter is a band-stop filter. This filter blocks all frequencies within a certain range and passes all others (Proakis & Manolakis, 2005). Since bird calls are characterized by a wide range of frequencies, the team had to be very particular in their choice of filters in order to prevent any information loss and to ensure that the filters are only removing unwanted noise.

The second signal processing technique the team used was discrete Fourier transforms (DFTs). Since the team was sampling analog signals from a microphone, all of the bird calls were digital signals. This means that some signal processing was done digitally using discrete signals. A DFT transforms one function in the time domain into another function in the frequency domain, and is defined below in Equation 1. The sequence of N complex numbers x_0, ..., x_{N-1} is transformed into the sequence of N complex numbers X_0, ..., X_{N-1} by the DFT according to the formula, where i is the imaginary unit and $e^{\frac{2\pi i}{N}}$ is a primitive Nth root of unity.:

Equation 1: Discrete Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}kn} \qquad k = 0, \dots, N-1$$

The transform is sometimes denoted by the symbol $\mathcal{F}$, as in $\mathbf{X} = \mathcal{F}\left\{\mathbf{x}\right\}$ or $\mathcal{F}\left(\mathbf{x}\right)$ or $\mathcal{F}\mathbf{x}$. In order to increase efficiency the team used a similar technique called a fast Fourier transform (FFT). An FFT is an efficient algorithm used to compute the DFT of a function. Using

FFTs instead of DFTs help improve performance and save run-time because the complexity of a DFT algorithm is $\Theta(N^2)$ while the complexity of a FFT is $\Theta(N \log N)$ ("Discrete Fourier Transform"). FFTs allowed the team to determine the frequencies contained within each bird call. This was beneficial because each bird call should have specific spectral characteristics that help differentiate it from other species. FFTs can also be easily performed in MATLAB.

Another signal processing technique the group utilized was cross-correlation. Cross-correlation is a measure of similarity of two waveforms, also known as a sliding dot product or inner-product. Similarly, for discrete functions, the cross-correlation is defined as**:**

Equation 2: Cross Correlation

$$(f \star g)[n] \stackrel{\mathrm{def}}{=} \sum_{m=-\infty}^{\infty} f^*[m]\ g[n+m].$$

Cross-correlation involves shifting one signal over another signal and looking for matches. It is similar to the convolution of two functions but instead of reversing a signal before multiplying and shifting it, correlation only involves multiplying and shifting (Cross-correlation). Cross-correlation allowed comparison of a given bird call with the database of bird calls. The bird calls with the highest correlation in the time and frequency domains are the most likely matches.

2. 3 Mel-frequency Cepstrum

The mel-frequency cepstrum represents the short-term power spectrum of a sound, which is based on a linear cosine transform of a log power spectrum on the nonlinear mel scale of frequency. The mel scale is a perceptual scale of pitches judged by listeners to be equal in

distance to one another. To convert hertz into mel, the popular formula shown below is used ("Mel Scale").

Equation 3: Conversion from Hertz into Mel

$$m = 2595 \log_{10}\left(\frac{f}{700} + 1\right) = 1127 \log_{e}\left(\frac{f}{700} + 1\right)$$

Mel-frequency cepstral coefficients (MFCCs) collectively make up a mel-frequency cepstrum. These MFCCs are derived from an audio file by first taking the Fourier transform of the signal. Then, the powers of the spectrum obtained are mapped onto the mel scale using triangular overlapping windows. The logs of the powers at each of the mel frequencies are taken and then the discrete cosine transform is taken of the list of the mel log powers as if it were a signal. The resulting amplitudes of the spectrum are the MFCCs. MFCCs are commonly used in speech recognition technologies as well as voice recognition technologies ("Mel-frequency Cepstrum"). Because of these uses, it could be believed that these MFCCs are useful in identifying species of birds. However, the MFCCs are very sensitive to noise, so a noisy bird song may not be well-identified.

Another consideration when using the MFCCs of a bird song or any other signal is that the first coefficient should be ignored. In some cases, the second coefficient should also have less consideration in the final result, as well. This is because the first coefficient represents the average power of the spectrum, while the second one represents the broad shape of the spectrum. The remaining coefficients represent the finer details of the spectrum and could be considered to be more useful features in identifying the spectra (Terasawa, 2005).

2.4 Bird Call Recognition

One challenge faced during attempts to identify actual bird songs is recognizing all songs made by a particular species of bird. A bird's call can contain a significant amount of information including the bird's species, sex, individual identity, his territorial and reproductive status, and his probability of responding aggressively or sexually to a potential recipient (Emlen, 1972). Due to the changes a bird may make to its calls to signal these messages, pinpointing one species of bird per call may prove difficult, though not necessarily impossible.

Douglas A. Nelson attempted to find variations among different species of birds using audio spectrograms. He found that birds could generally be differentiated not by the comparison of the spectrograms themselves, but by the analysis of ten acoustic features, such as range of frequency or song duration, derived from their audio spectrograms (Nelson, 1989). This information could help the group greatly since it may not be necessary to correlate the spectrographs to ones in the team's collections, but can instead analyze them and compare their attributes. This may be more successful because many species of birds have a few different songs they may use for different situations (Byers, 1995). The only drawback is determining what features would differ enough to identify a variety of birds independently.

Some scholars differentiate bird calls into two categories: undirected songs (UD) and female-directed songs (FD) (Sakata et al., 2008; Byers, 1995). In a study on Bengalese finches, Sakata et al. found some differences, such as song length and syllable repeatability, between UD and FD songs. The team should take this into account when determining their variables to identify species of birds, since when a bird is calling a mate, he may change his song. An example of this could be when a bird makes a territorial, long-distance call or a call to a nearby potential mate. The differential attenuation, reflection, and absorption of sound could cause the

bird to change his song when making a long or short-distance call (Konishi, 1970). However, Emlen believes that songs or repertoires of songs of a species of birds must have different qualities than those of other species in order to avoid species misidentification (Emlen, 1972).

Another consideration the team should account for is that birds in different locations may have different calls. Research suggests that bird calls of same-species birds can differ from those only a few miles away (Byers, 1995). This could make it difficult for the team to test their system, particularly if the team uses pre-recorded signals from other regions. In order for the team to get accurate measurements for the signal analysis, the group may need to analyze multiple samples for many different regions and determine what variables do not change substantially.

With this project, the team hoped to overcome the major obstacle of varying bird calls from the same species. This has been a barrier in the field of ornithology for quite some time and was one of the largest challenges of the team's project. After that challenge was faced, the goal of this project was to create a system for bird call identification that could be utilized without an expertise in signal processing.

Chapter 3: Methodology

The purpose of this section is to describe the methods that the group used to complete this project. This section contains a scope of the project, system block diagrams, algorithm creation and testing and design component choices.

3.1 Scope

The goal of this project is to be able to identify a bird species based on its call. The objectives that were necessary in this endeavor are as follows:

- Resolve a way to identify one bird species despite a variation of calls from that species
- Analyze and identify bird species using MATLAB
- Analyze and identify bird species using the C language
- Implement the C program on a digital signal processing chip

The team began by resolving a way to identify one bird species despite a variation of calls from that species. To do this, the literature was researched to find a way to isolate that species from all other species based on certain characteristics of the bird call. The team elected to utilize the mel frequency cepstral coefficients as described in the background chapter. The specific details of this implementation will be discussed later.

Next, this processing stream was coded in MATLAB. Many of the necessary functions were available online, and credit for the mel bank code goes to Mike Brookes (Brooks, 2009) while credit for the MFCC function goes to Olutope Foluso Omogbenigun (Omogbenigun, 2009). The team's contribution to this code was correlating the resulting MFCCs from one bird

call with another and plotting the results. Results for this portion of the project can be found in the next chapter.

After that, the group implemented the same processing stream using the C language. This code was written mostly from scratch although some of the functions were adapted from the TI DSP library. The platform selected for algorithm implementation was a Texas Instruments C6713 DSP development board (or DSK) because this hardware was readily available to the group and the group had previous experience with it.

After the C code was complete, a significant amount of time was spent debugging it. Although the code compiled, there were fundamental errors in the processing that needed to be resolved. A detailed account of these errors can be found in the obstacles section of the next chapter.

3.2 System Block Diagram

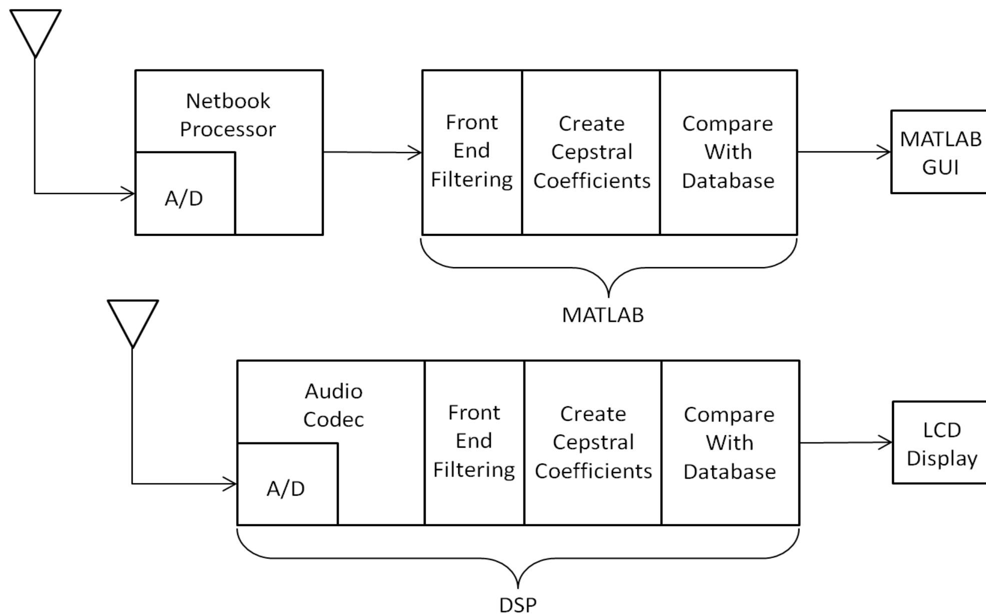

Figure 1: System Block Diagrams

The above block diagram describes the intended processing stream for both languages of the project. Although the team was unable to implement an LCD display, the team did manage to fully implement an MFCC based bird song identification algorithm in C. Currently, the output of the algorithm is displayed on the console window of the DSK's development environment. For now, this prevents the device from being easily used in the field.

The analog input the system receives from the microphone is converted through an AIC23 codec on the DSK and then is put through a lowpass filter. The lowpass filter has a passband up to 10 kHz and a stopband that begins at 11 kHz. Some birds' songs do go above this range, but it is rare for them to as the average bird song frequency is 4 kHz (Brand, 1938). Testing the bird song signals of interest confirmed that a 10 kHz passband was sufficient for the purpose of the team's samples. There was no highpass filter because a highpass filter would remove the low frequency envelope information, which is part of the information represented by the MFCCs.

Next, the MFCCs were found to identify the species of bird. To find the MFCCs, the Fourier transform is first taken of a portion of the signal. Then, the power of the spectrum obtained is mapped onto the mel scale and take the logs of the powers at each of the mel frequencies. Finally, the discrete cosine transform of the list of mel log powers is taken and the amplitudes of the resulting spectrum are the MFCCs. A more detailed description of this process is discussed later.

By comparing these MFCCs to MFCCs calculated from training data collected for each bird species, the algorithm can identify the bird species. For this project, these trained MFCC collections were generated using at least three bird calls from four different bird species and use

the same MFCC method described above. These MFCC replicas formed the four specie database.

From there, the system outputs the closest three matches to the console, complete with how well the database MFCCs correlated with the MFCCs of the incoming signal. If the correlation is below 0.50, the particular database signal is not considered a match. This prevents a signal that does not correlate well with any signal in the database from being incorrectly identified.

3.3 Creation of an Algorithm

In order to ascertain whether or not the final product would work as intended, some research and testing had to be done. To test the algorithm before beginning the final product, MATLAB was used. The team created a test algorithm in MATLAB to find out what would and wouldn't work when identifying bird calls with the MFCCs. The test algorithm had to take in two waveforms, the number of MFCCs needed, and the frequency at which the waveforms would be sampled. The two waveforms would be compared based on the separate MFCCs created for each signal. For the MATLAB implementation, the team was lucky enough to find a piece of code written by Olutope Foluso Omogbenigun (Omogbenigun, 2009) which was available for public use. The group also used part of the MATLAB Signal Processing Toolbox, VOICEBOX, for the project (Brookes, 2009).

The MFCC algorithm assumed that the signal vectors were the same size. To get the signals to the same size, the signals were checked for size and then padded if necessary. Then, for the signals to be compared easily, the most common features had to occur at the same time for a correlation function to work. In order to get two bird songs to occur simultaneously in both

signals, the time domain signals are cross-correlated, and the signals are zero-padded accordingly to align them in time.

The MFCCs are calculated using the method described in the Mel-frequency Cepstrum section of this report. These MFCCs are then cross-correlated and the higher the resultant correlation values, the more likely the two signals will be considered a match.

3.4 Obtaining Samples

In order to test the algorithm, the team needed to obtain samples. The largest library of high quality bird sounds is found within the Macaulay Library, Cornell's Lab of Ornithology. As a short term solution, the team was able to obtain the Macaulay Library's commercial CD of bird songs titled, "An Evening in Sapsucker Woods", through the Gordon Library at WPI. This library contains thirty samples of bird songs from different species. However, to successfully test the algorithm the team needed multiple calls from the same species. After contacting Macaulay Library Director Mike Webster, the team was able to obtain many samples from the northeast. Since the song of the same species can vary greatly from region to region and the team obtained samples from all over the north east, the team was able to effectively test the algorithm with these samples.

After sixty bird song samples were obtained from Macaulay Library at Cornell University, the database could be created. The team specified that they needed ten bird calls from six species of birds – the Cape May Warbler, Magnolia Warbler, Mourning Warbler, Carolina Wren, House Wren, and Sedge Wren. The team wanted to compare birds within the same families, but also within the same genus. It could be thought that birds within the same genus or even family could have similar bird calls and may be confused with each other. To test this, the team chose the three warblers of the Parulidae family and three wrens of the Troglodytidae

family. Within these groupings, the team chose to have two birds of the same genus and a third bird of a different genus for the warbler group and three different genera for the wren group. In the warbler group, the two birds of the same genus are the Magnolia Warbler and the Cape May Warbler.

In order to further identify each bird type, the group listened to all ten songs of each bird species and categorized them aurally. Some songs were difficult to categorize. In the end, most bird species had three to four distinct categories for their songs and one or two outliers. After categorizing the songs aurally, the group used the algorithm to test their hearing. Songs that sounded the same did, in fact, correlate into the same groups as expected, with a correlation of at least 75%. Songs in different types for the same bird species correlated anywhere from 40% to 80%. Interestingly, some of the outliers seemed to be mixtures of songs from other categories. For example, a bird could begin a song with half of a certain song type and end it with half of a song of another type. Furthermore, the samples that the team received were not devoid of noise. In fact, many of the samples had other birds singing in the background. One important note is that the bird song identification algorithm identifies the loudest bird song in the recording. The figure below depicts a signal with noise being correctly identified by the algorithm. Results from this MATLAB testing can be found in the results chapter and the Appendix A.

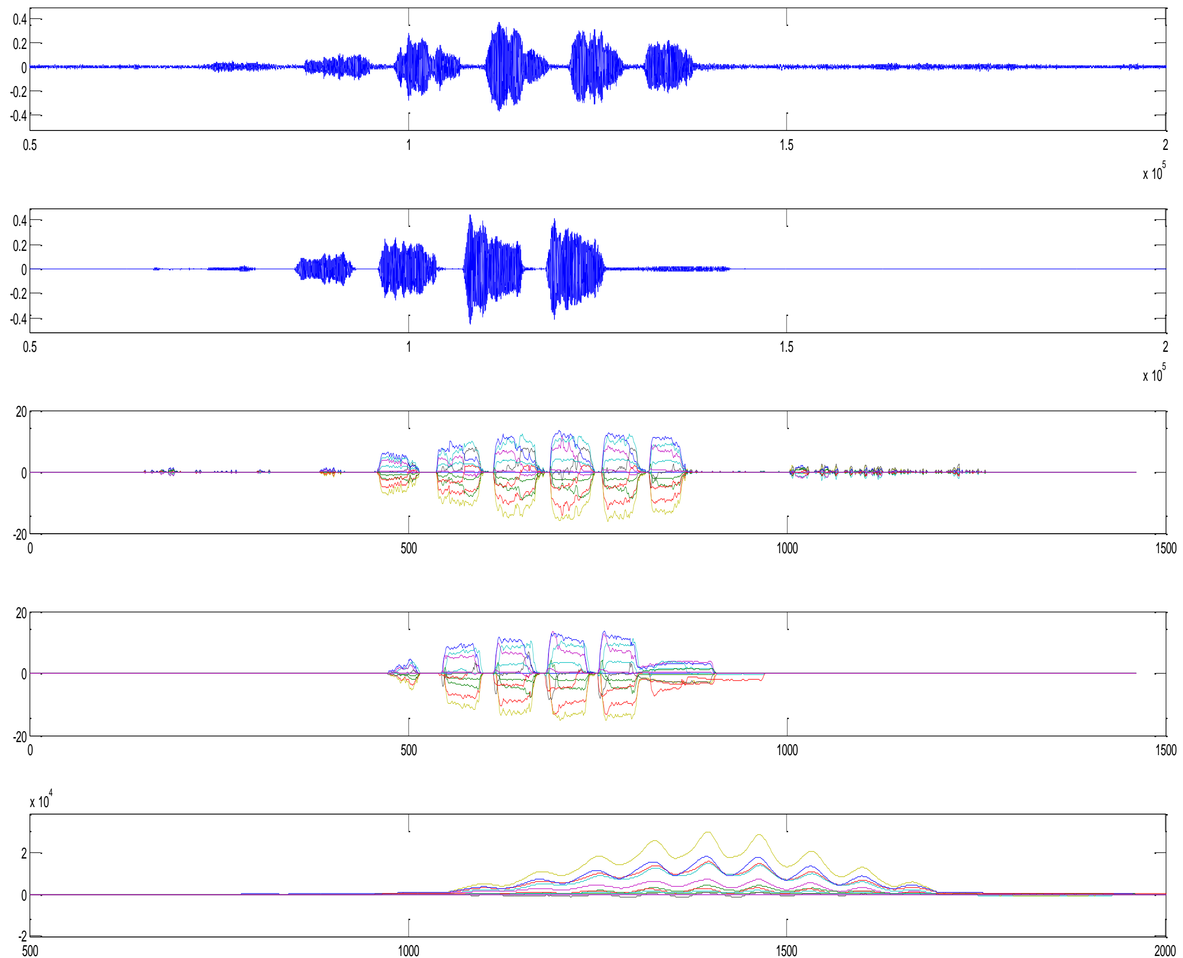

Figure 2: Cape May Warbler Song Comparison With Background Noise in Signal 1

3.5 Modular Design Choices

The following sections describe each specific design choice. Each component was chosen for many reasons and specifically fits the application. The components described are a netbook processor, front end filters, digital signal processing chip, LCD screen and microphone. The team looked at netbooks initially considering that a simple system realization could run the algorithm in MATLAB on a netbook and if time permitted the MATLAB code could be

converted to the C language and at that point MATLAB would no longer be required to run the project.

3.5.1 Netbook Processor

The team examined netbooks with two criteria in mind, performance requirements and cost. The chosen netbook, the Asus Eee PC netbook with Intel Atom Processor, was as inexpensive as possible while still fitting the team's performance requirements. This netbook only costs $279.99 and can still run MATLAB with ease. Table 1 below shows the system requirements to run the 32-bit version of MATLAB.

Table 1: System Requirements for 32-Bit MATLAB

Operating Systems	Processors	Disk Space	RAM
Windows XP Service Pack 2 or 3	Intel Pentium 4 and above	680 MB (MATLAB only)	512 MB (Recommend 1024 MB)
Windows Server 2003 Service Pack 2 or R2	Intel Celeron		
Windows Vista Service Pack 1 or 2	Intel Xeon		
Windows Server 2008	Intel Core		
Windows 7	Intel Atom		
	AMD Athlon 64		
	AMD Opteron		
	AMD Sempron		

The chosen netbook comes standard with Windows XP Service Pack 3 and runs on an Intel Atom processor. It contains a 160 GB hard drive and 1 GB of DDR2 memory. All of these system requirements meet or exceed the requirements necessary to run MATLAB. This netbook also contains an integrated sound card, which is required to record the bird songs. Even with all these requirements, this netbook is cheap, which is why it was chosen for the initial system's development ("System Requirements – Release 2010a").

3.5.2 Front End Filter

In order to reduce the noise associated with input bird songs, the team needed to implement some filtering after they acquired their samples. The MATLAB Filter Design and Analysis Tool was used to create an FIR Direct-Form I Equiripple Filter. The magnitude response of the filter is seen in Figure 4 below. It is a stable filter of order thirty.

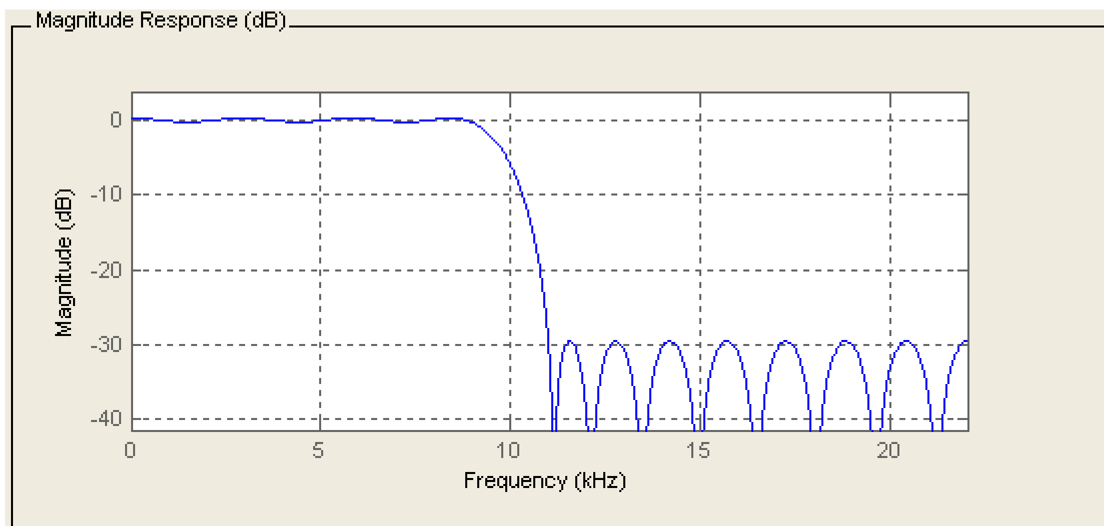

Figure 3: Magnitude Response of Lowpass Filter

In order to implement this filter in C, the coefficients were exported as single precision floats into a C header file. These coefficients were used to implement this filter on the DSP chip. Floating point math was used instead of fixed point math in the C filter to increase stability and ease of implementation. As each sample was recorded, it was filtered in real time to avoid additional signal storage problems.

3.6.3 Digital Signal Processing Chip

The digital signal processing chip that was chosen for this application is the Texas Instruments TMS320C6713. It was chosen for many reasons, first being ease of access and use. This chip and its development environment are used in WPI labs for other courses and are

readily available to the team. The second reason it was chosen was because all team members have previous experience programming this chip. This chip is a floating point DSP, which makes it easier to code than a fixed point DSP chip. The C6713 also has an AIC23 stereo codec, which is ideal for audio applications. It can sample at multiple rates between 8-96 kHz, which fall within the frequency ranges for bird songs. The chip also has line in and out as well as a microphone in and headphone out, which is necessary for this application. Lastly this board has a USB interface to a PC, which will make it easier to program. Figure 2 below shows the block diagram of the C6713 DSK (Chassaing, Rulph, 2005).

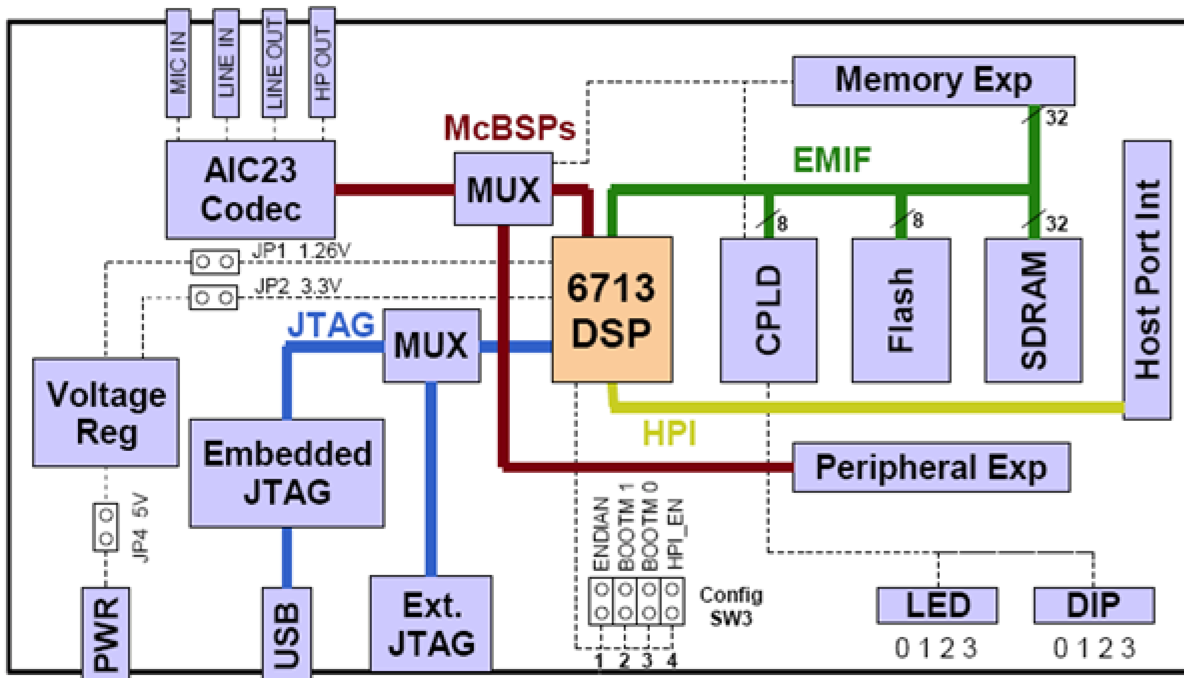

Figure 4: Block Diagram of C6713 DSK

3.6.4 LCD Screen

The LCD screen that best fits the application is the LCM-X12232GXX from the Epson SED1520 Series of LCD drivers. This LCD screen is from a family of dot matrix LCD drivers that are designed for displaying characters and graphics. The application requires the team to be

able to display the names of birds that are potential matches and this LCD will allow the team to do that with ease. The team also has considerable experience with both character and graphics display for this LCD from prior course work. The LCD is also readily available because it is used for other courses. Note that the team did not implement this LCD screen due to time constraints, but given additional time the group could have.

3.6.5 Microphone

When this project is implemented as a handheld device, the team decided that it should be up to the user to decide the quality of microphone that they want to use based on their application. Microphones can range in cost from $20 to over $500 depending on the quality and range the user wants. Since the team is testing on high quality recordings of bird sounds, the group decided to use an inexpensive microphone. The microphone that bests fit the project budget is the Audio-Technica Unidirectional Microphone. The team chose a unidirectional microphone because it records sounds coming from one direction, rather than using an omni-directional microphone which would record potentially many sounds coming from all directions. The MFCC algorithm functions best when there is the least amount of undesired noise, and a unidirectional microphone would help minimize the noise. Lastly, this microphone was chosen because it is the least expensive microphone found that would not hinder the performance of the device.

3.7 Algorithm Implementation

This section explains the team's design choices and why each decision was made. It is true that much of the team's methodology came from the literature, but there is a reason behind each suggestion that came from the literature. Although the team used the processing stream from the MATLAB code as a template, the bulk of the C implementation was written from

scratch with the exception of Texas Instruments' speed-optimized FFT functions ("TMS320C6713 DSP Starter").

3.7.1 Filtering and Frame Generation

First of all, the MFCC algorithm takes a five second signal in the time domain sampled at 44.1 kHz and separates it into 861 overlapping frames containing 512 samples. This means that the step size is 256 samples, exactly half of the frame size. This was chosen in order to prevent more than two frames from having duplicate data, which in turn keeps the same data from correlating across many frames. The overall algorithm works by recording a large buffer of 220500 samples and pulling a 512 sample frame out of the buffer. Each frame then moves through the processing stream and when one frame is complete, the next frame moves through the processing stream. A flow chart of the complete process is shown below in Figure 5.

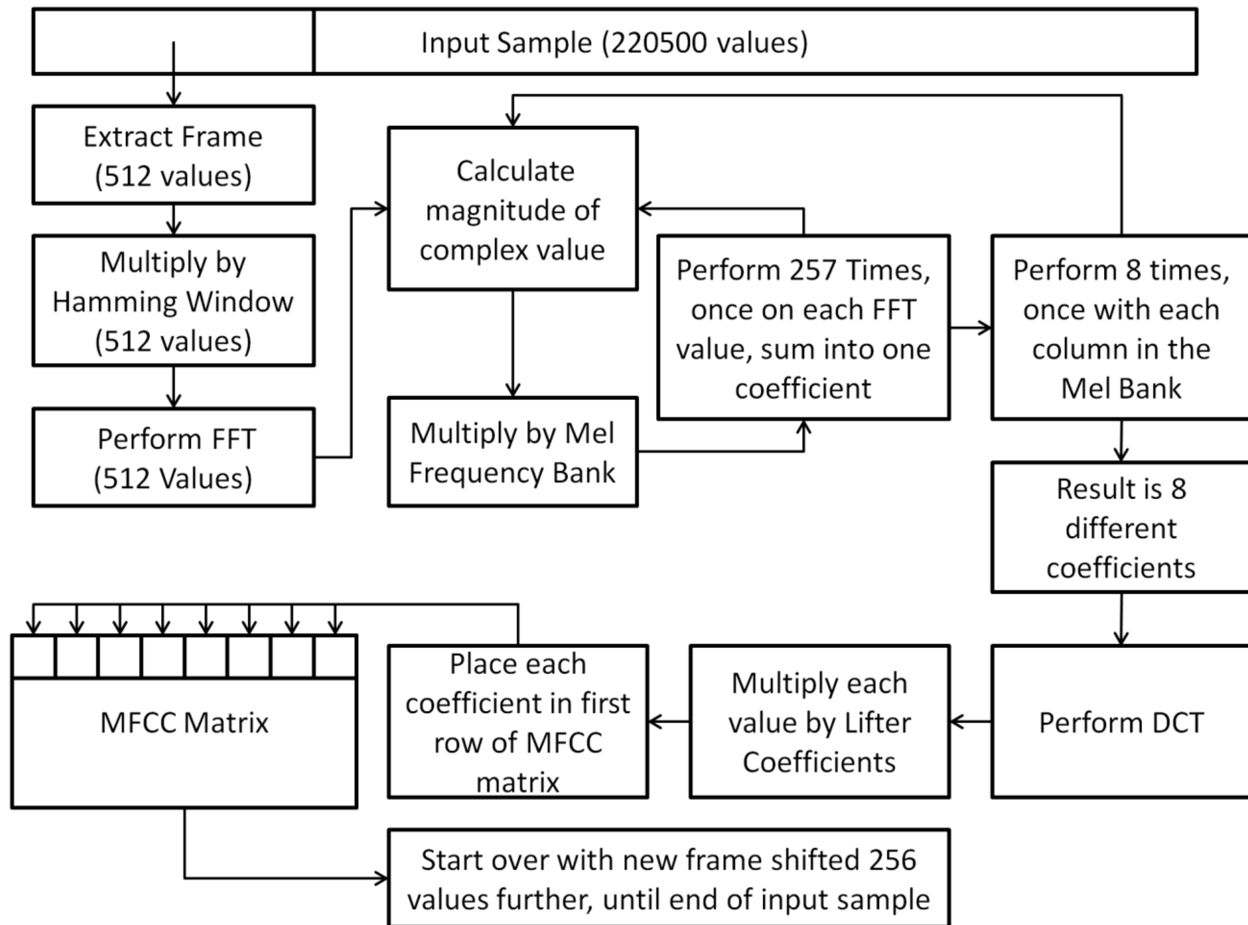

Figure 5: Algorithm Flow Chart

The team implemented a lowpass filter as the first step of the processing stream to remove excess noise and prevent aliasing. As previously mentioned, the filtering is performed in real-time to reduce storage requirements. Although the overall algorithm does not run in real time, a short computation time is still desired as the user may not wish to wait until the bird flies away before discovering the identity of the bird.

3.7.2 Windowing Concepts

From there, each frame was multiplied by a 512 point Hamming window. This causes smearing in the frequency domain although it smoothly tapers the signal towards a zero-value at

point 0 and N-1. There were many window functions to choose from such as a Bartlett window, a Hanning window, a Hamming window, and a Blackman window, but a Hamming window was chosen because it maintained an adequately wide main lobe without having too great of a side lobe amplitude. The equation for a Hamming window is depicted below:

$$W[n] = 0.54 - 0.46\cos(\frac{2\pi n}{N-1}),$$

where n is the particular point being calculated on the Hamming window and N corresponds with 512, the length of the Hamming window.

3.7.3 Mel-Scale Filtering

At this point an FFT is taken of each 512 sample frame. The result is then mel-scale filtered. This means that the signal is filtered in the frequency domain with a mel frequency bank.

This bank is an independently generated matrix full of coefficients and it depends upon sampling rate, desired number of mel frequency cepstral coefficients, and the number of points in the FFT. The mel frequency bank was separately implemented as a matrix (or array of arrays) of constants generated in MATLAB. As mentioned above, the only variables necessary for the creation of this matrix are to remain constant regardless of the input signal. Therefore, declaring a constant matrix as opposed to calculating this matrix each time the program runs saves processing time. The implementation creates a sparse matrix with eight columns (corresponding with the desired number of MFCCs) and 257 rows (corresponding with half of the samples in the frame) of coefficients that are multiplied by the complex-valued FFT output.

The FFT output frames are then multiplied by corresponding points in the mel frequency bank and then summed together. For example, the first FFT output frame has 512 samples. Due

to the even symmetry of the FFT, the algorithm only needs to consider half of the FFT output. The algorithm multiplies the first 257 samples within the frame by the 257 corresponding coefficients in the first row of the mel frequency bank. It then sums each of these products together into one data point. This data point represents one frequency-filtered frame for the first coefficient. The process is repeated eight times for each of the eight MFCCs in the mel frequency bank. At this point, the process repeats itself with the next frame. The result is a matrix of eight columns representing the eight MFCCs and 861 rows representing each frame that originated from the buffer. This method was chosen in order to compress the data while still retaining a sufficient pool of results. A flow chart of the above frequency domain filtering process is shown below in Figure 6.

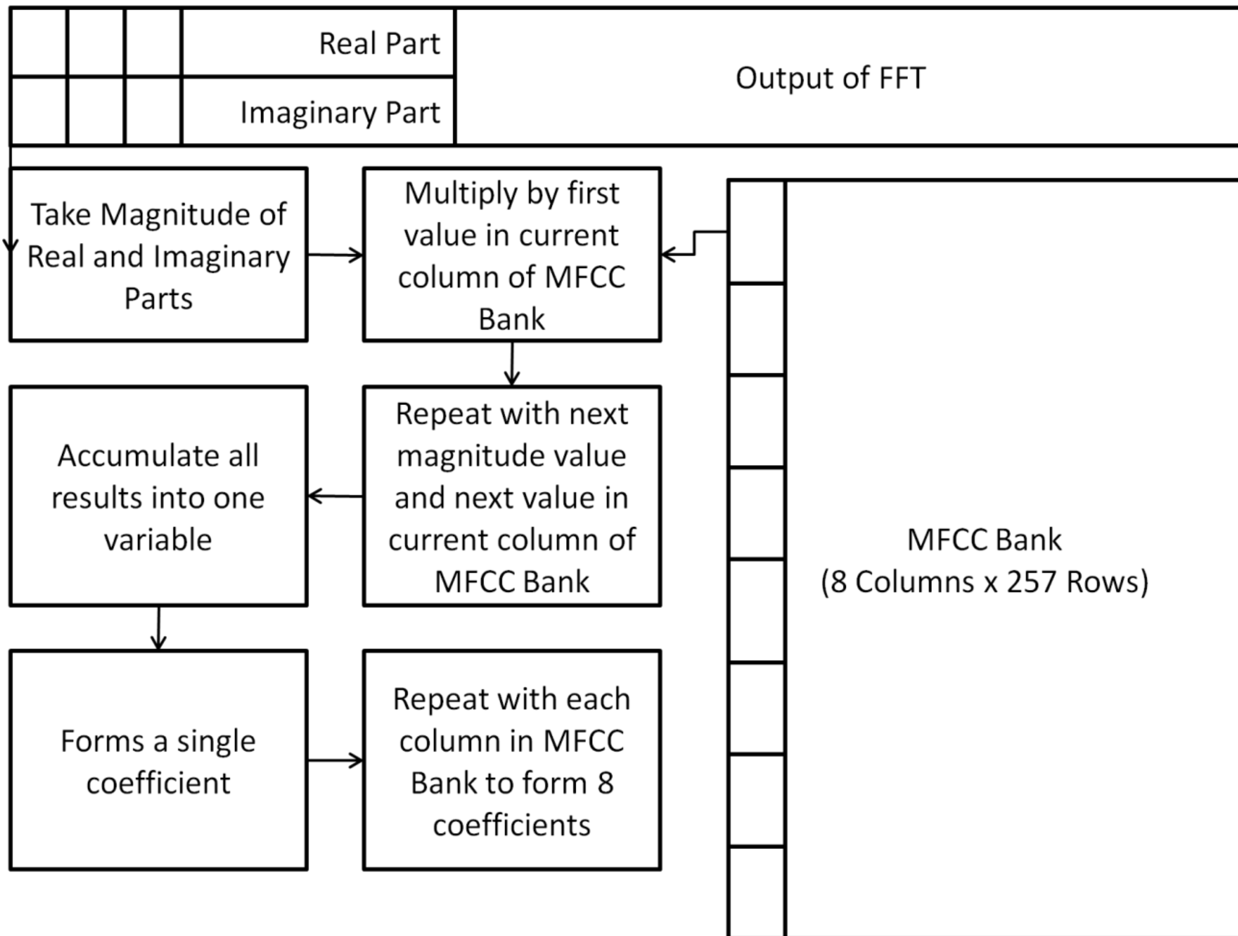

Figure 6: Frequency Domain Filtering Flow Chart

MATLAB results returned 24 MFCCs, but it was determined through testing that eight coefficients displayed an adequate set of features and the remaining 16 MFCCs were ignored. Therefore, only eight MFCCs are utilized in the C implementation to increase the speed of the algorithm.

Before the final matrix is assembled, however, the logarithm base ten is taken of the data points. This returns the data to a linear frequency scale. The discrete cosine transform is then taken of each column of mel logarithmic powers. The results are eight time domain signals, one for each MFCC. These signals correlate well with other similarly processed signals of the same species.

3.7.4 Correlation and Database Implementation

To quantitatively measure the similarity of two of these resulting matrices, the data was correlated from two different bird call signals. One resides in a database within the program, and the other is the recent input signal that has been processed. The first MFCC from one signal is correlated with the first MFCC from the other, and this process is repeated for each of the eight MFCCs. Through testing the team determined not to take the first MFCC into account because it measures the power of the signal, which frequently correlates with many other bird call signals as the signal power can be similar from species to species. Additionally, it may poorly correlate with a bird of the same species that happens to be further away from the microphone. Without taking this first coefficient into account, the average of the remaining seven coefficient correlations often made it possible to determine the bird's species.

When determining whether two signals are from the same species, one can only say with a certain level of confidence whether or not the signals match. The higher the correlation, the more likely the two birds are of the same species. Either one is confident that two signals match or one is unconfident that two signals match. It was determined that an appropriate correlation cutoff is 0.50. This is realistic because of the nature of the implementation. It searches through a database of different bird songs, so if none of them can come up with a 0.50 average correlation across the MFCCs then it is determined that the signal is from an unknown source. It is possible that a given signal matches just over 0.50 with the database, which is not a strong correlation, but the system is designed so that the signal eventually finds a better match within the database.

The database is implemented as a lookup table. If one database entry correlates well and becomes the new top match, the prior second place match overwrites the third place match, the prior first place match overwrites the prior second place match, and the new first place match

overwrites the prior first place match. This allows the output data to contain both strings and floating point numbers in order to give the user meaningful information.

Chapter 4: Results

This section will outline the results of the project including MATLAB testing results, C testing results, obstacles encountered along the way, and other general data pertaining to the project.

4.1 MATLAB Testing Results

It was necessary to test the MFCC algorithm to be sure that it would function in the intended manner. To do this, a variety of bird songs were sampled and compared with the algorithm. To start out simply, two signals compared were from a single Carolina Wren. As seen in the figure below, the MFCCs of each bird call had many similarities. The correlation match-up for these bird calls was around 85%.

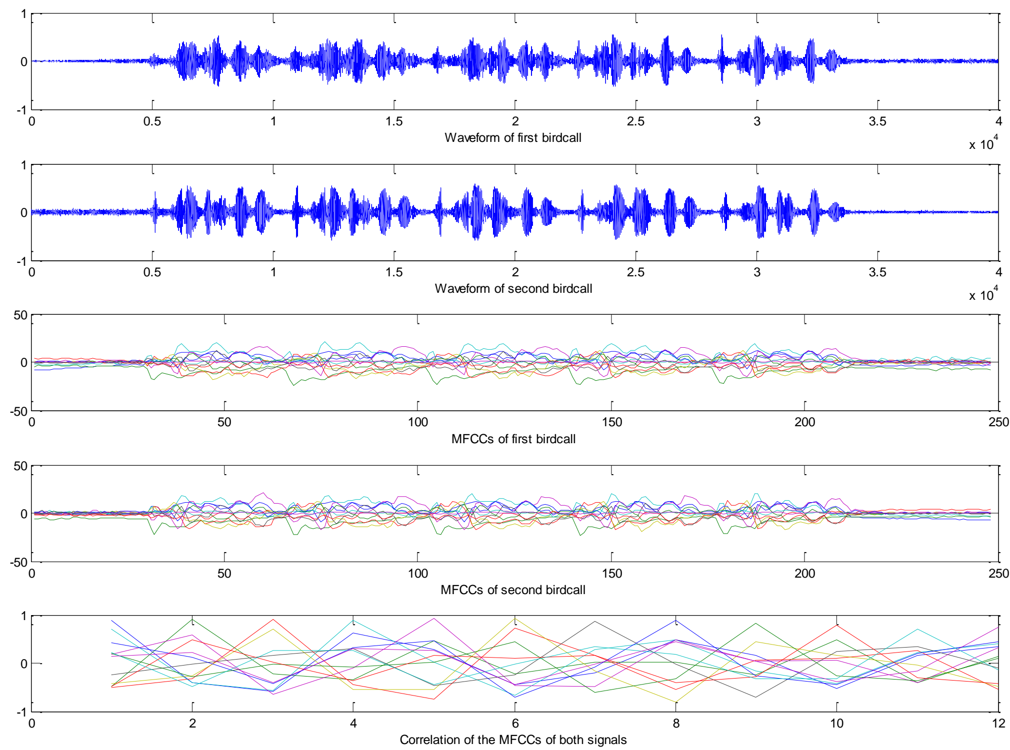

Figure 7: Comparison of Two Calls from the Same Carolina Wren

It was necessary to test this algorithm with signals that are vastly different from bird songs. Therefore, the algorithm was also tested with two human whistle trills. The correlation of these signals had an average correlation of MFCCs around 88%. The comparison of these two whistle trills is shown below.

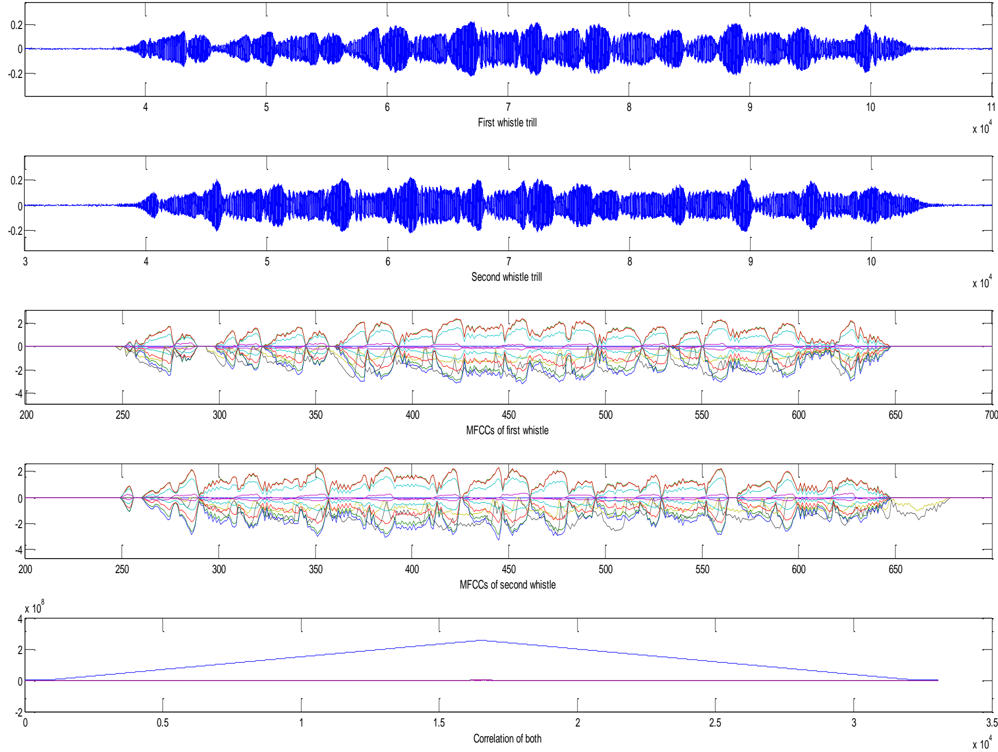

Figure 8: Comparison of Two Human Whistle Trills

After identifying that the algorithm was reasonably reliable at identifying bird songs, the MATLAB algorithm was automated for further testing. One song was taken from each bird song type of each bird species and added to the database in order to decrease the processing time of the identification algorithm. The automated algorithm took in only one bird song as well as the number of MFCCs needed and the sampling frequency. The algorithm then compared the unknown bird song with all of the songs in the database.

The algorithm was then tested with bird songs that were not in the database to see how accurate the algorithm was with signals unknown to the system.

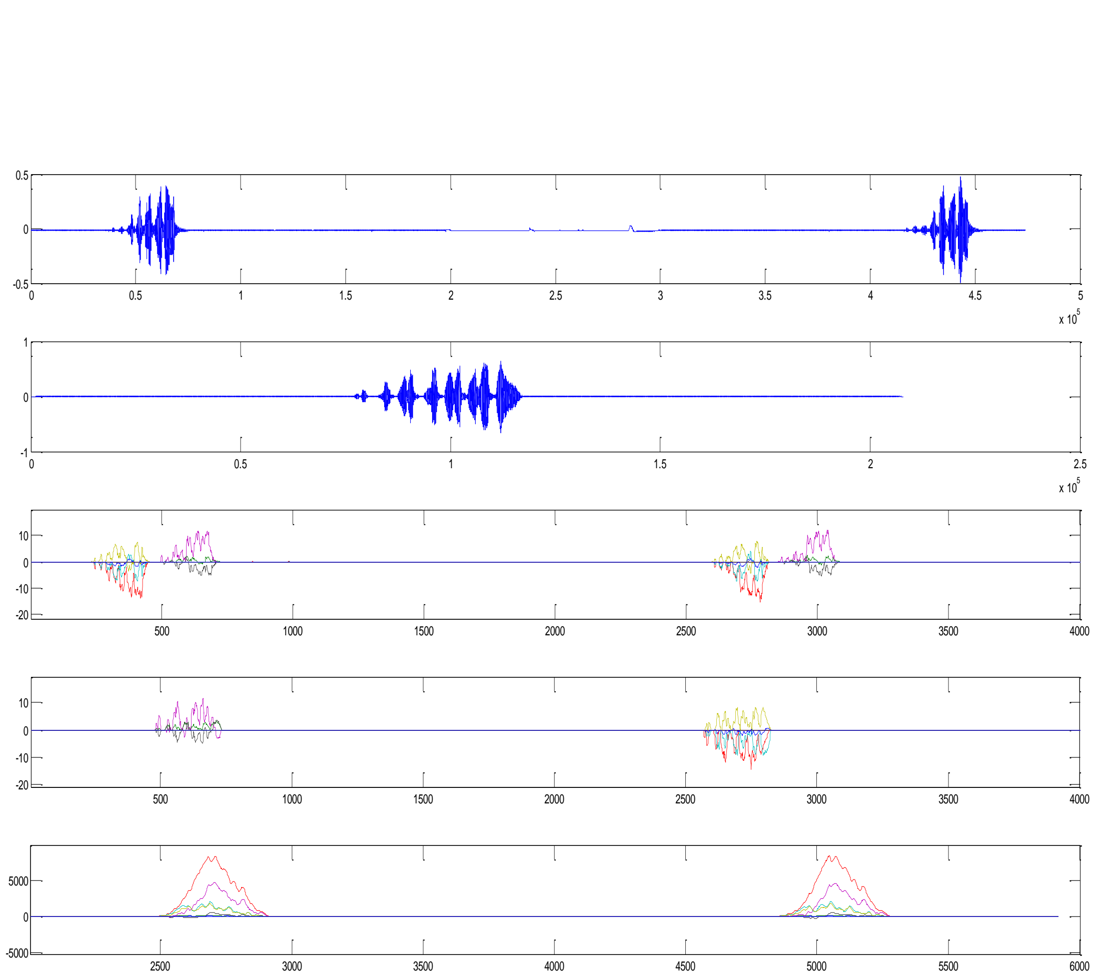

Figure 9: Magnolia Warbler Comparison with One "Unknown" Song

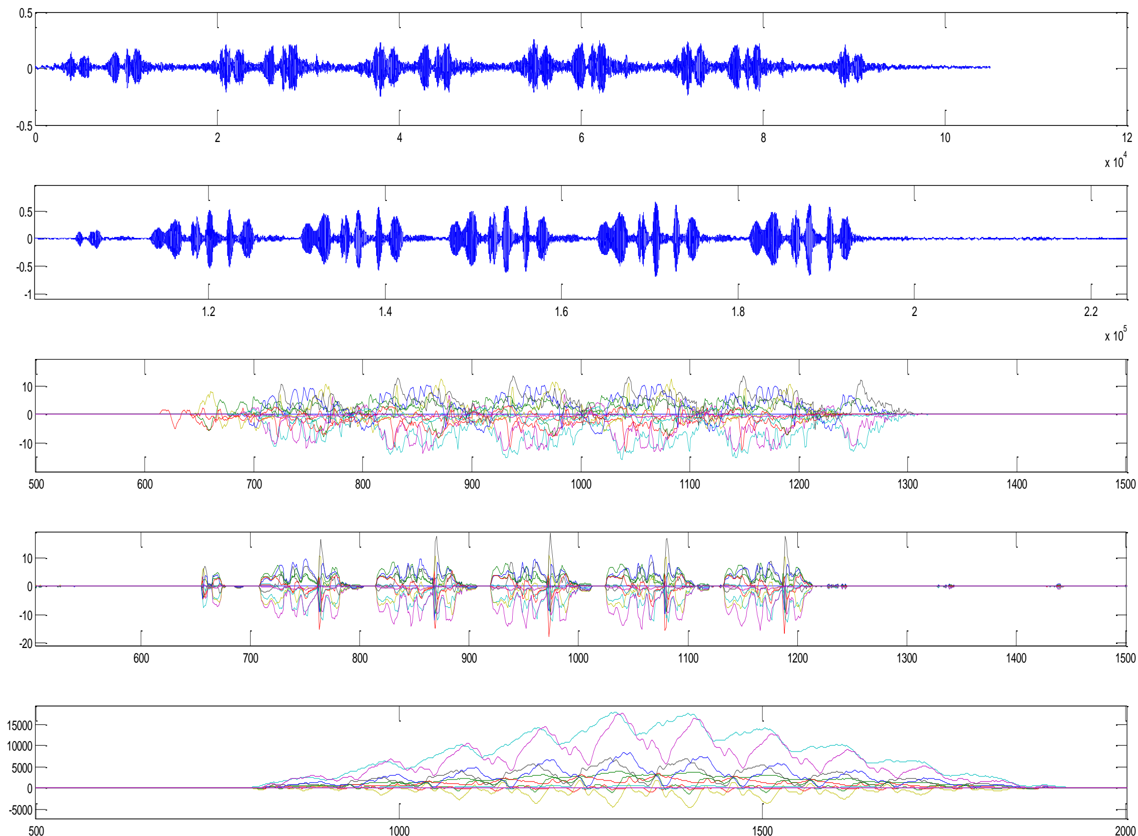

Figure 10: Carolina Wren Comparison with One "Unknown" Song

The available songs from the Macaulay Library that were not part of the database were tested to find the accuracy of the algorithm for a small sample set. For this sample set, the algorithm was 93.75% accurate in identifying bird songs. However, although the algorithm correctly identified the birds most of the time, some bird calls would not correlate well with the same species. For example, the House Wren's "song" is for the most part just a series of tweets. When creating the algorithm, it was not a consideration that the term "bird song" may mean something very different from one species to the next. The results of the MATLAB experiments are shown below.

Table 2: Bird song average correlations and percentages choosing the correct bird

Bird Type	Cape May Warbler	Magnolia Warbler	Mourning Warbler	Carolina Wren	Overall
Sample 1	0.8655	0.5989	0.7042	0.7115	
Sample 2	0.8326	0.7324	0.9782	0.5139	
Sample 3	0.7941	0.7158	0*	0.6178	
Sample 4	0.787	0.9387	0.7031	0.5455	
Sample 5	0.9382	0.9415		0.6178	
Average Correlation	**0.84348**	**0.78546**	**0.795166667**	**0.6013**	**0.756351667**
Percentage Correct	**100**	**100**	**75**	**100**	**93.75**

*A zero on this table indicates that an incorrect bird species was chosen by the algorithm.

4.2 Testing the Algorithm in C

This section outlines how the group tested each part of the algorithm in C to prove that it worked correctly. The parts of the algorithm listed in this section are Fast Fourier Transform (FFT), the Discrete Cosine Transform (DCT), the correlation function, Hamming window function, and MFCC algorithm.

4.2.1 Fast Fourier Transform Test

The team tested the FFT by using a 2 kHz input signal and performing an FFT on this data. The plot of the input data can be seen below, as it was recorded by the development board using a frame buffering function and interrupts.

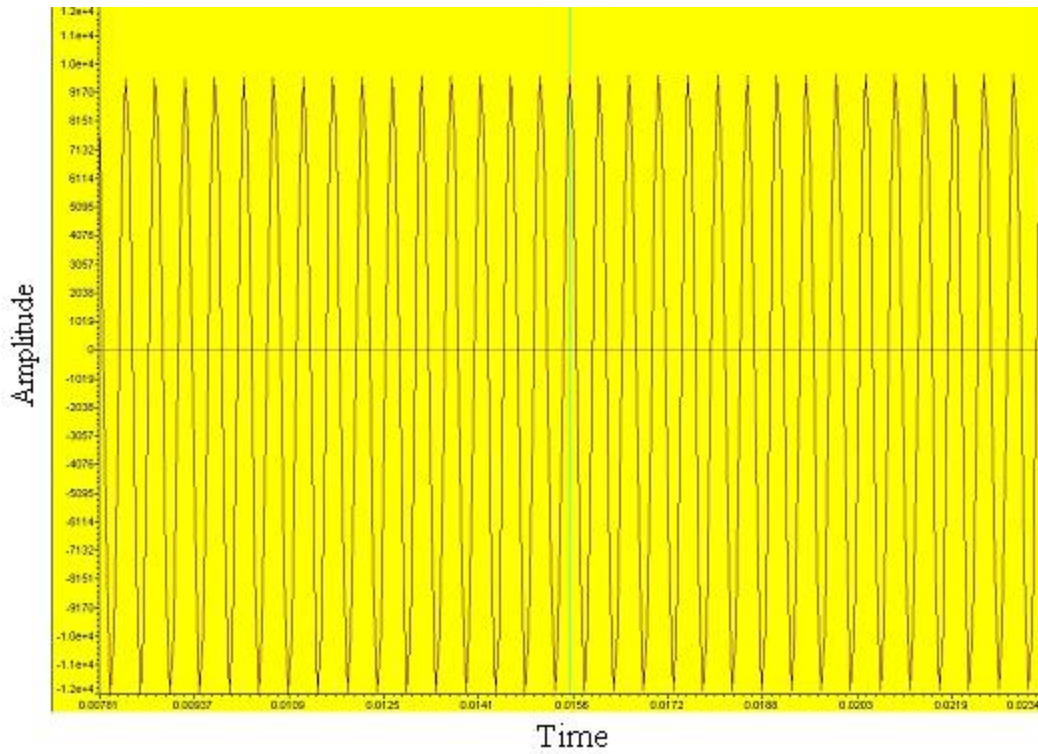

Figure 11: 2 kHz Sine Wave

The group performed an FFT on this data to prove that it functions appropriately and the results are shown in the graph below.

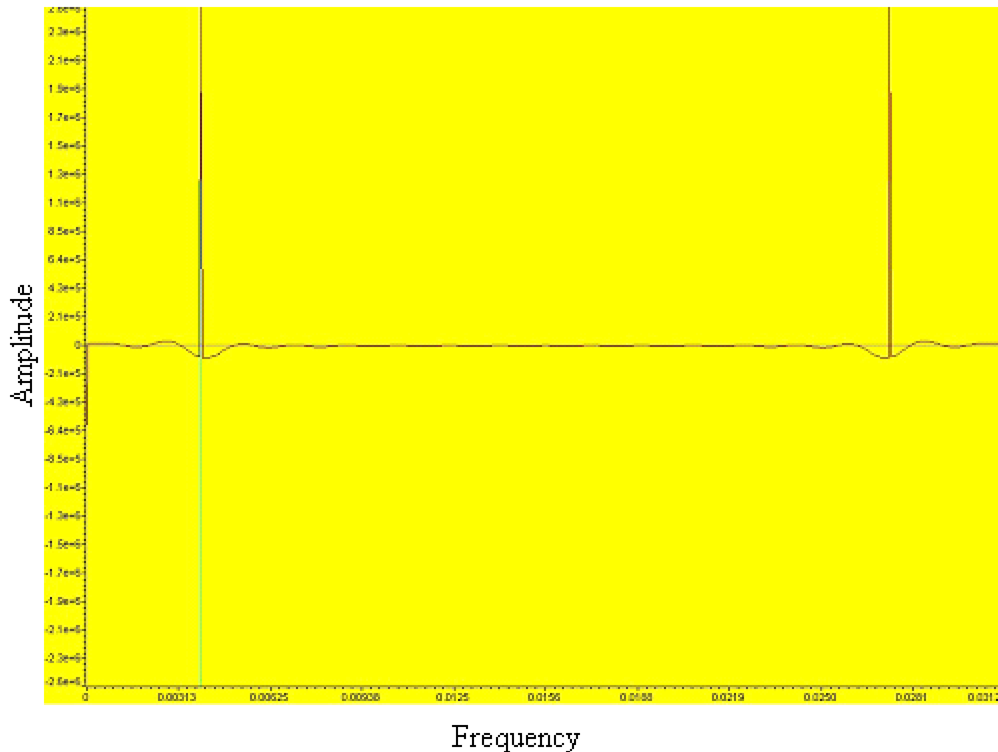

Figure 12: Fast Fourier Transform of 2 kHz Sine Wave

As shown, there are two clear spikes at the appropriate frequencies. Because the DSP chip was sampling at 16 kHz, the Nyquist frequency was 8 kHz, causing the FFT to wrap around and making another spike appear at 6 kHz, which is expected. The team eventually increased the sampling frequency, but this module was proven to work beforehand so it was not retested.

4.2.2 Discrete Cosine Transform Test

The discrete cosine transform was tested using a small array of predefined values. The output was compared to the output of the DCT function in MATLAB to prove that both functioned in the same manner. The input samples were an array of eleven values from 0 to 1, incrementing by 0.1 each time. The graph of the DCT of this data is shown below.

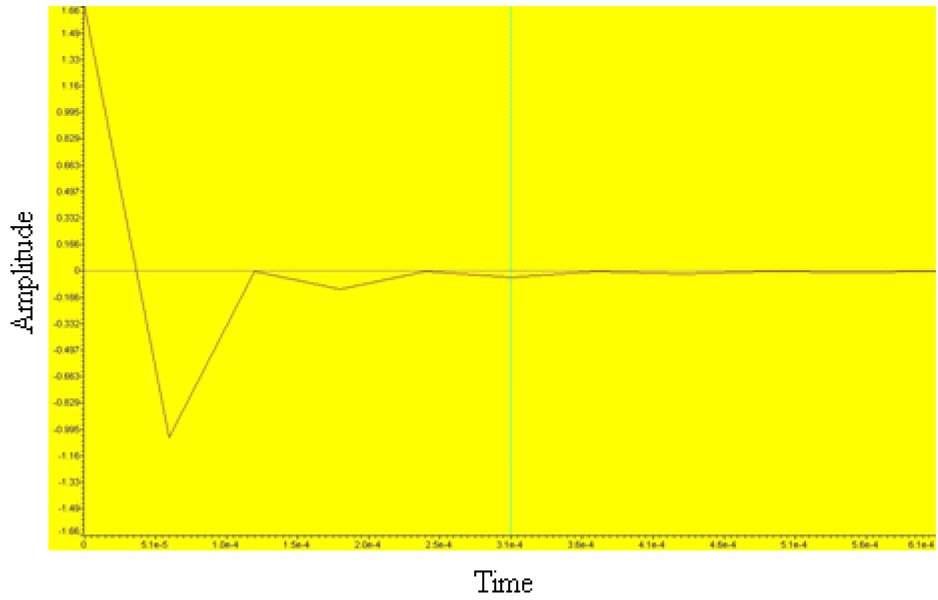

Figure 13: DCT of Input Samples in C

The same input samples were used in MATLAB. The resulting graph of MATLAB's DCT is identical to DCT calculated in C, and can be seen below.

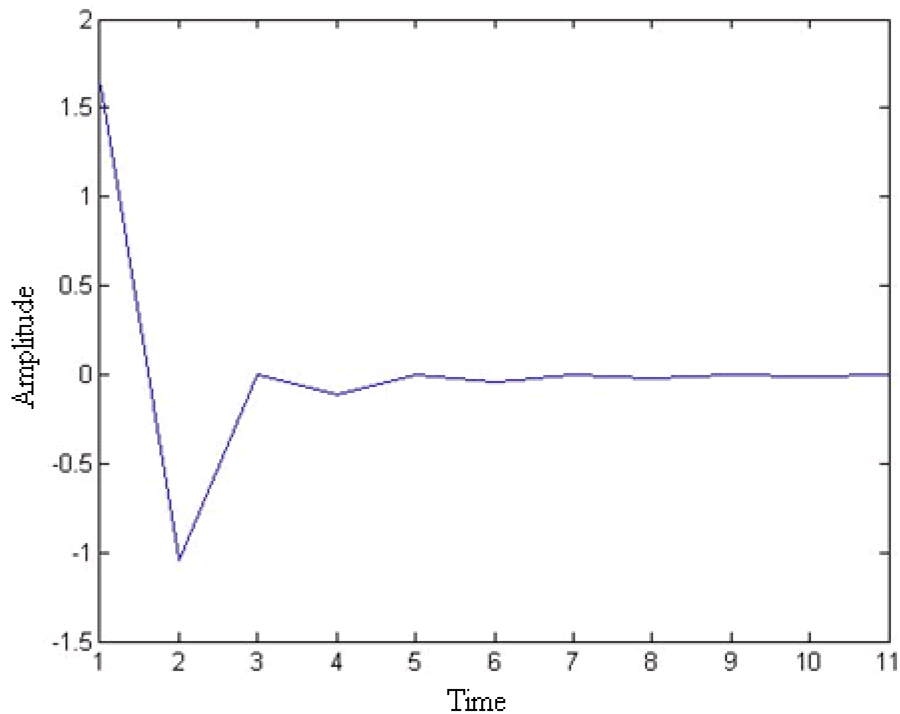

Figure 14: DCT of Input Samples in MATLAB

Both of these graphs are clearly the same, proving that the team's DCT implementation in C functions exactly the same as the DCT in MATLAB.

4.2.3 Correlation Function

The next function that the group tested was the correlation function. The team used a wrap around cross correlation function, which should produce a 100% correlation with two different signals, even if they are shifted in time, while a rectangular windowed cross correlation function would not produce a 100% match with two signals shifted in time. The wrap around cross correlation function was compared to a non wrap around function using two identical time shifted signals. The two signals that were used in the correlation can be seen below.

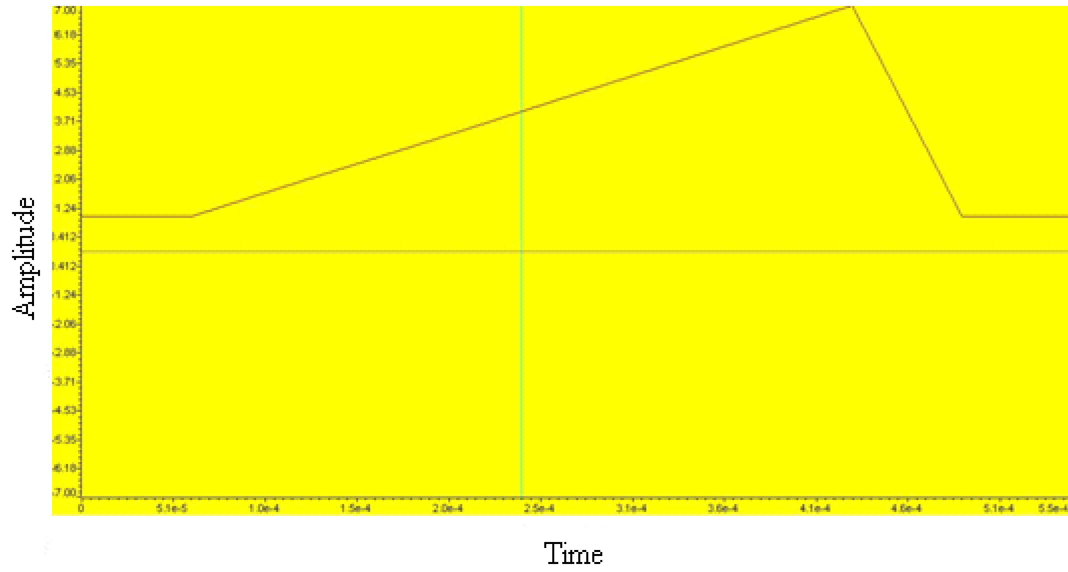

Figure 15: Original Input Signal

The next signal is the time shifted version of the signal shown above.

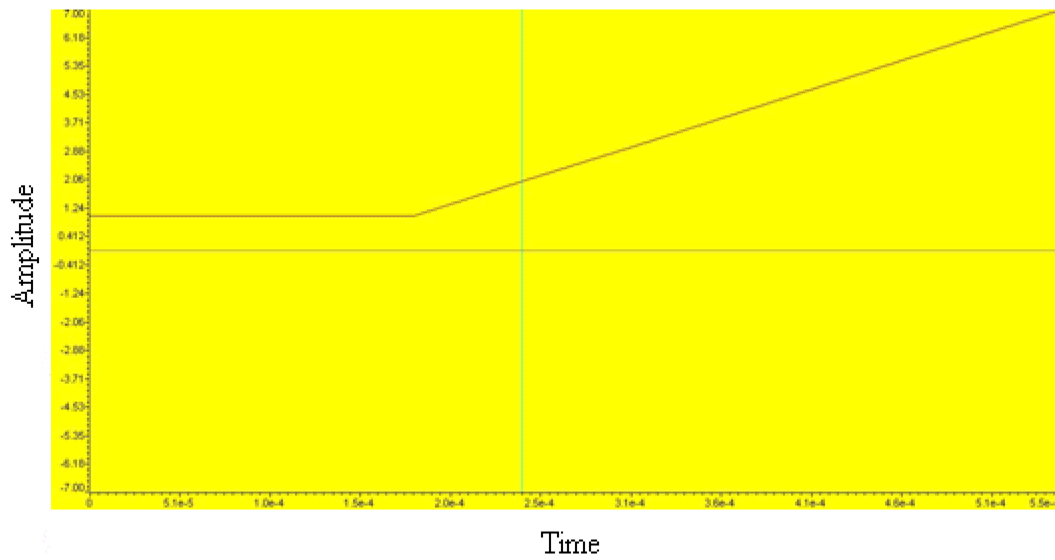

Figure 16: Time Shifted Input Signal

The rectangular windowed correlation function returned a match of 81%, a graph of the cross correlations at each point in the time shift can be seen below and the highest correlation is visible as the highest spike in the graph.

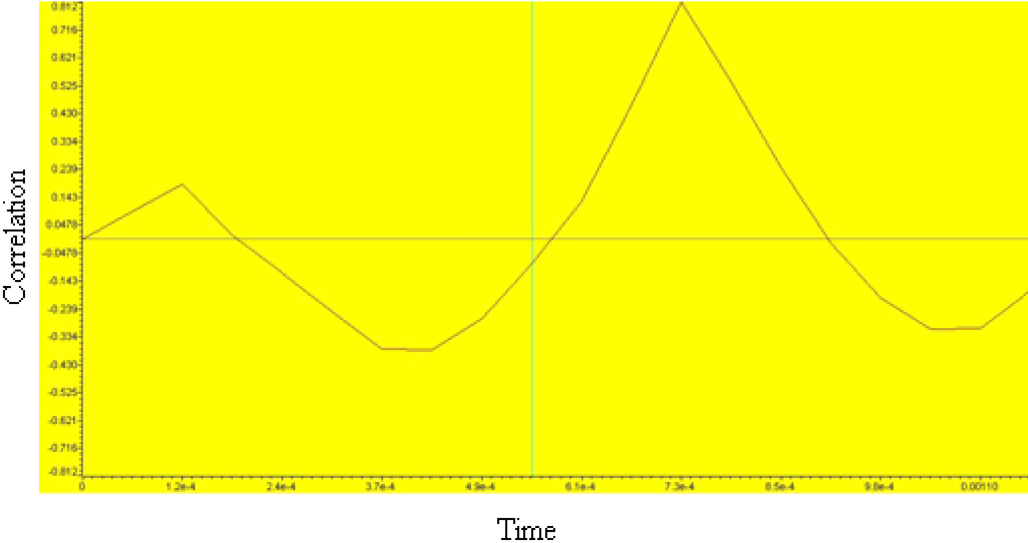

Figure 17: Rectangular Windowed Cross Correlation of Input Signals

The next graph shows the wrap around cross correlation function used in this project. It is clear that this function achieves the 100% match as expected.

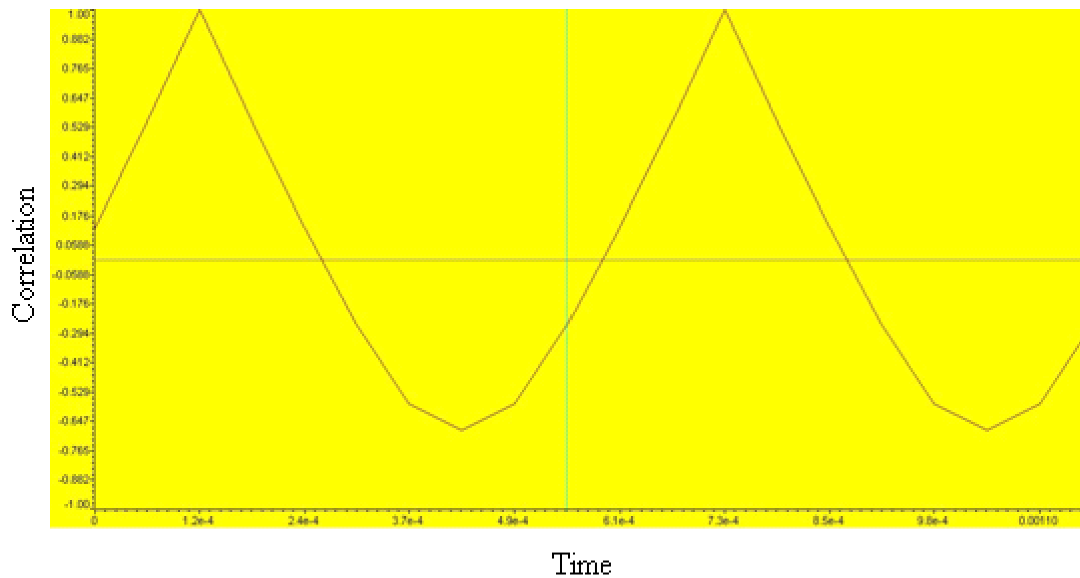

Figure 18: Wrap Around Cross Correlation of Input Signals

Unlike the previous graph, the team's function successfully achieves a full match, proving that the wrap around cross correlation function successfully returns a 100% match on time shifted signals, while the windowed cross correlation function does not.

4.2.4 Hamming Window Function

The last function in the algorithm that the team tested was the Hamming window function. They tested to make sure that the hamming window of any input size would produce the correct result. This was shown by testing against the hamming function in MATLAB that the team used in their MATLAB algorithm. To test the function, the group used a Hamming window of size one hundred and compared the graphs from the MATLAB and C functions. The first graph below is the hamming window function in C.

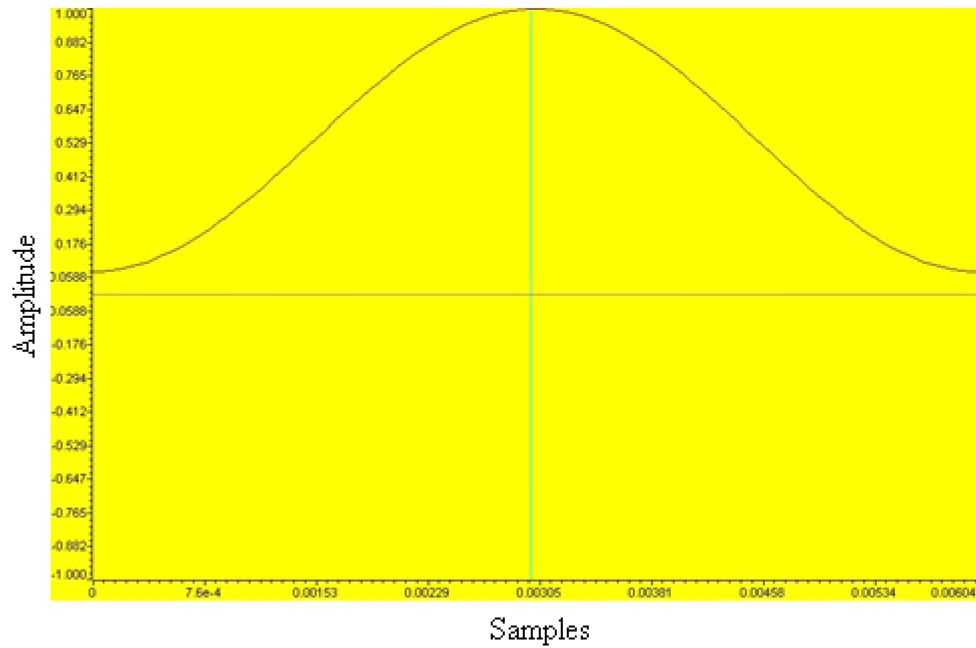

Figure 19: Hamming Window Function in C

This graph shows a smooth Hamming window curve of one hundred, exactly similar to the MATLAB curve shown in the graph below.

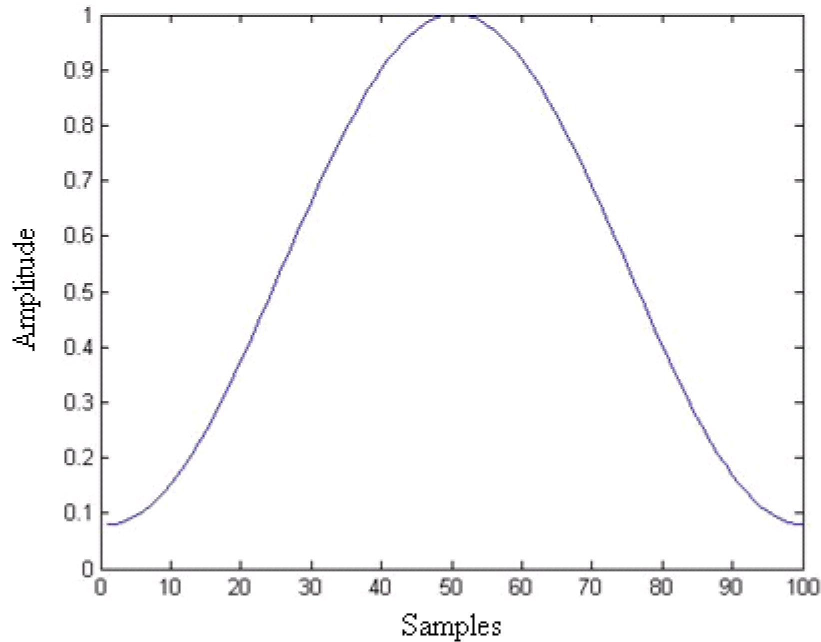

Figure 20: Hamming Window Function in MATLAB

This verifies that the team's hamming window function in C operates exactly the same as the hamming window function in MATLAB that is used in the MATLAB version of the algorithm. The values range from just under 0.1 to 1 in each case with a smooth curve from 0 to 100.

4.2.5 Testing the MFCC Algorithm

To test the final algorithm, the group played bird signals through a computer's sound card, and recorded them onto the development board via the line in audio input. There is a four song database in the C code, and each recording is compared against the four birds and the one with the highest match is returned along with the correlation percentage. The average correlation between the bird songs and the library was 73.379%, which is about ten percent lower than the average correlations for MATLAB, but that is expected because of the amount of noise in the signal in the C program. The correlations for each bird can be seen in the four tables below. Each

bird song was played five times and the correlations with each song in the library are displayed. Correct matches are shown in green while incorrect matches are shown in red.

Table 3: Cape May Correlations

Cape May Warbler Correlations					
	1	2	3	4	5
Cape May	0.791672	0.769434	0.776941	0.82797	0.764354
Carolina	0.775264	0.677315	0.748241	0.699389	0.708077
Magnolia	0.702732	0.685081	0.654663	0.690685	0.725959
Mourning	0.700666	0.725249	0.642847	0.77486	0.75083

Table 4: Carolina Wren Correlations

Carolina Wren Correlations					
	1	2	3	4	5
Cape May	0.778909	0.706168	0.754237	0.772692	0.769016
Carolina	0.723266	0.77087	0.709941	0.779205	0.778964
Magnolia	0.645363	0.702268	0.683869	0.6588	0.767653
Mourning	0.717673	0.717515	0.690208	0.675831	0.755142

Table 5: Magnolia Warbler Correlations

Magnolia Warbler Correlations					
	1	2	3	4	5
Cape May	0.750675	0.766816	0.676857	0.766021	0.678284
Carolina	0.7297	0.683459	0.733553	0.76459	0.688087
Magnolia	0.676445	0.79717	0.772399	0.63541	0.730769
Mourning	0.755648	0.748219	0.730483	0.657862	0.708104

Table 6: Mourning Warbler Correlations

Mourning Warbler Correlations					
	1	2	3	4	5
Cape May	0.704017	0.745095	0.776211	0.674686	0.776768
Carolina	0.745325	0.675282	0.730211	0.713834	0.75301
Magnolia	0.787587	0.781311	0.81613	0.77414	0.737939
Mourning	0.796035	0.749148	0.792085	0.787375	0.78637

The algorithm picked the correct bird fourteen times out of twenty attempts for an overall percentage of 70%, although the percentage ranged from 60% to 100% depending on the species of bird. The MATLAB algorithm was over 90% accurate, however we stated that due to noise, our algorithm would be less accurate than the MATLAB version. However the group believes an accuracy of 70% is still acceptable for a proof of concept project and considers this algorithm to be successful.

4.3 Obstacles

The purpose of this section is to outline many of the obstacles that the team encountered when implementing the bird call identifier on a hand-held device. A brief list of major problems that were solved is the record and playback function, the front end filtering, imaginary numbers, the Fast Fourier Transform, memory mapping, and noise.

4.3.1 Record and Playback Function

The first problem that the group encountered was implementing a record and playback function that never had to be reset. This function would allow the user to press a button to start recording a sound, then allow the user to play back the sound with another button press. The first attempt that the team made was through the use of interrupts. The decision was made to write an interrupt service routine to collect samples at the designated sampling frequency of 16 kHz. Interrupts were only enabled when the record button was pressed, and then disabled when the device was finished recording its preset time of five seconds. However this did not prove to work because the interrupts were never successfully enabled therefore no samples were ever recorded.

In order to fix this problem, the team took a different approach. The decision was made to move away from interrupt service routines and use polling to take the samples. With this method, the sampling frequency was set to 16 kHz and a loop was used to poll five seconds of samples into a buffer after the button was pressed. The program then moved into a playback stage where it would await user input via a different button to playback the recording. This method seemed to prove successful and the group was able to successfully record samples, store them in a buffer, and play them back to the user. Note that the group later implemented a 44.1 kHz sampling frequency in order to obtain a better quality input signal.

4.3.2 Front End Filtering

The second problem that the team encountered was adding front end filtering to the samples before performing any calculations on them. Typical bird calls lie in the range of 2 kHz to 8 kHz. The group decided to implement a bandpass filter to pass frequencies within this range in order to avoid any aliasing. The first problem encountered was that the team was unable to set any filter parameters beyond 8 kHz because the sampling frequency was 16 kHz. This means that the high frequency cutoff for the filter cannot start at 8 kHz and end by 9 kHz, it must end at 8 kHz, which means the cutoff must start lower than 8 kHz. In order to avoid filtering out any data from the sample, the group decided to make a higher order, very steep filter. However, a bandpass filter with these parameters created an unexpectedly high order filter. When the team implemented this filter into the code, the filtering took a very long time because of the high order. To solve this problem, the team decided to implement separate lowpass and highpass filters instead of a single bandpass filter. The lowpass filter stops frequencies around 7.8 kHz and the highpass filter stops frequencies below 2 kHz. This allowed both filters to be lower order and

actually improved the overall filtering speed. The team later changed the sampling frequency to 44.1 kHz in order to obtain a higher quality input signal.

4.3.3 Complex Numbers

The next problem the group found was consistency with imaginary numbers. When MATLAB performs the required algorithms, it automatically handles complex numbers. However, in C code, the floating point data type does not account for complex numbers, and the programmer needs to create its own complex number structure to keep track of imaginary parts. The team realized that there were imaginary numbers while coding the Fast Fourier Transform function in C. An FFT always outputs both real and imaginary numbers, so the team needed to implement complex number structures. After stepping through the MATLAB algorithms one at a time, the group learned which functions dealt with complex numbers and which sections used only real numbers.

After realizing that certain sections of code were now using complex arguments, the team had to double check to see if every function still worked correctly. The group was using the absolute value function from the C math library, but this function was not working because it was intended for double data types rather than complex numbers. In MATLAB, the absolute value function abs() accepts complex arguments, calculates the magnitude of the complex number and returns a real number. After figuring this out, the team wrote their own function to calculate the magnitude of complex numbers, which returned a floating point real number to emulate the MATLAB algorithm.

4.3.4 Fast Fourier Transform

The Fast Fourier Transform (FFT) takes samples in the time domain and transforms them into the frequency domain. The FFT is an integral part of the algorithm and it will not work without a fully functional and accurate FFT. After initial testing of Texas Instrument's optimized radix-2 FFT, the team could not get a working result. The group was testing the algorithm using Code Composer Studio (CCS) and graphing the output buffer of the FFT. The first mistake that the team was making was with CCS's graphing functions. The group was using the FFT Magnitude graphing function, which produces its own FFT on the data selected and graphs it for the user. This means that the group was graphing an FFT of an FFT, which clearly produces the wrong results.

After realizing how to properly test the FFT, the group continued to run into more problems. MATLAB's FFT was performing a 400 point FFT on the frame, which is the entire frame size. An FFT is ideally performed on buffers that are only divisible by a power of two in length. Some FFTs take longer to perform the calculation when the number of points is not a power of two, and some do not work at all. MATLAB's FFT was able to make this calculation, however it takes much more time to calculate a 400 point FFT than a 512 point FFT. After more debugging, the team discovered that TI's optimized radix-2 FFT only works when the number of points is a power of two. The team solved this problem by changing the step size and frame window size so that each frame is 512 points long. The FFT now performs a 512 point FFT, solving the problem.

4.3.5 Memory Problems

TI's TMS320C6713DSK has a limited amount of onboard memory. Ideally, the team would implement all of the code and variables to be contained in the fast RAM (the IRAM),

however the amount of space is limited, only 512kB. The DSK also contains a large amount of slower RAM (the SDRAM), about 16MB, but running code or variables from this memory will greatly increase the amount of time necessary to run the algorithm. Lastly the chip contains FLASH memory, however it is very difficult to program and it was recommended that the group avoided using the FLASH memory at all costs.

While writing the software, the team encountered several memory problems. The search algorithm works by storing many large two dimensional arrays in memory, called f-matrices, and compares the incoming signal to these arrays. The group knew it was necessary to store these f-matrices in the SDRAM, however the program would not compile while trying to store these large arrays into external memory. The group realized that chip writes constants into a buffer called ".cinit", which is located in the fast memory, and this buffer was filling up if more than one f-matrix was written into slow memory. The team figured out how to edit the DSK's command file which designates where each section of memory is located. This problem was solved by relocating the ".cinit" as well as the f-matrices buffer into the SDRAM.

After solving the ".cinit" memory problem, the team quickly ran into another. Now that the f-matrices could be successfully written into external memory, the group could write their search algorithm to correlate the incoming signal's f-matrix with the stored library of f-matrices. After finishing this algorithm, the team ran out of code space in the IRAM. The first attempt at fixing this problem was to move all code memory to SDRAM. This allowed the code to compile correctly, however the program now took around ten minutes to perform all the MFCC calculations and correlations on a single recorded sample. It is realistic that the device does not have to operate in real time; however a delay time of ten minutes is very unreasonable.

The team noticed earlier in the project that the chip does not allow the user to access all of the slow and fast memory. After some research, the group realized that many sections of the chip are reserved for certain functions and others are simply set to "Protected." CCS allows the user to change the memory mapping of the chip, as long as the user does not overwrite any of the reserved sections. The group realized that they could expand the upper address of the IRAM from 0x0002FFFF to 0x00080000, more than doubling the amount of usable space, without overwriting any reserved sections. This allowed the team to keep the code memory in the IRAM and successfully expand the code without running into any more memory problems.

4.3.6 Noise

A major problem with MFCCs is noise. The MATLAB algorithm works with high quality bird samples from the Macaulay Library that contain minimal noise. However, when the DSK implementation recorded samples to create f-matrix libraries, as well as when recording incoming samples, there is a significant amount of noise present. The lowpass filter catches the majority of the noise outside the passband; however a significant amount of noise is still present. The group has made many attempts at solving this problem. The first attempt was to use MATLAB's f-matrices that contain limited noise. The problem that occurs is the MATLAB algorithm uses a different frame size and step size, so that the correlation between the two is mediocre. In order to solve this problem, the team would have to change the MATLAB algorithm and recalculate each bird sample.

The next attempt was to use recorded bird songs played from the computer into the DSK. This produces a high level of machine noise and produces mediocre calculations as well, with correlation percentiles around 50 to 60 percent. Also, expected high correlations are not much higher than expected bad correlations and sometimes it is difficult to distinguish them from each

other. In order to clean up the f-matrices, the group tried to implement a cutoff value with each f-matrix coefficient. This essentially minimized the value of insignificant parts of the MFCC in an attempt to isolate the actual information in the MFCC. This produced significantly higher correlations, around 90%, however every correlation was then around this level and it was still very difficult to distinguish which correlations are supposed to match and which are not. The last method the team looked into is to compare which songs have higher matching coefficients instead of taking simple averages of the coefficients. For example Bird A may be the correct match; however one of its coefficients had a very low correlation rate, which is dragging down its overall correlation. Bird A may have higher correlations in almost every coefficient; however because of the single poor correlation its average may be lower than that of Bird B, the incorrect match. The original algorithm would return Bird B, however this algorithm would realize that Bird A was higher in every coefficient but one, therefore it is the better match. Testing proved that this method was ineffective. This noise issue could be solved in any future work that would be attempted after this project.

Chapter 5: Conclusions

This chapter will summarize the group's project and draw conclusions based on the results. It will also recommend future work that the team has suggested in order to create the intended final product.

5.1 Discussion of MATLAB Results

The MATLAB results were excellent for their purpose. It was possible to determine a bird's species with a high degree of confidence given the small sample set. Further testing is necessary to determine how well the algorithm is able to handle very large databases containing more than six species, but the team believes the MATLAB results can be used as a valid proof of concept.

5.2 Discussion of C Results

The C results were sufficient. For some species it is not possible to claim with confidence which species the bird call originated from. Further research is necessary to alter this MFCC algorithm to become more robust in the presence of noise.

However, the C results highlighted that the algorithm is sensitive to noise. As the team tested each module within the algorithm, the results were comparable with the MATLAB results. Only when the final MFCCs are generated and then correlated together is there a discrepancy. Furthermore, when the final blocks are tested with constant, known vectors, they correlate appropriately. This evidence suggests that the MFCC algorithm may not be robust in the presence of noise.

5.3 Future Work Recommendations

This section will explain the team's suggestions for device improvement. This includes a broader scope for each facet of the project.

5.3.1 MATLAB Future Work

As described above, the database that began in MATLAB needs to be significantly larger. It is possible that the MATLAB results change as more birds are added to the database. The algorithm may struggle identifying two similar species that are not currently present within the database. Furthermore, it is likely that as the bird songs within the database increases, the probability for multiple songs to match the input signal increases. Results that prove the algorithm is still valid for a database with a larger number of bird songs would confirm that the MFCCs are indeed identifying the unique differences between signals and classifying them accordingly.

5.3.2 C Implementation Future Work

The C implementation needs work in two major areas. First of all, the algorithm needs to modify its implementation such that it is not as strongly affected by noise. This may entail creating additional algorithms. For instance, another algorithm might have better classification features for a given signal, such as linear discriminate analysis (LDA).

Additionally, the C implementation will need to be optimized for speed. As the algorithm grows in complexity, the processing time will increase. There are many ways to do this. One could implement an algorithm that analyzes the signal and determines which of the potentially many classification algorithms to pass an input signal to. One could also implement a way to input a general geographical region to prevent comparison against irrelevant database samples.

These suggestions are possible solutions in addition to general C optimization techniques and practices such as hand-optimized assembly and automatic CCS optimization settings, and there may be a tradeoff against memory consumption, which is another obstacle.

5.3.3 Hardware Design Future Work

The hardware design part of this project was largely abandoned due to time constraints, but there is a considerable amount of work to be done. First of all, the output of the system needs to be changed to a more portable solution. The cheapest and easiest way to do this would be to implement an LCD screen. More functions are required in the C implementation to support more than a simple stdout printf call. The dot matrix LCD screen discussed earlier in the component choices section would be the team’s choice, however LCD driver circuitry would be necessary in addition to the LCD screen.

The hardware also needs to be reworked. The TI C6713 is relatively inexpensive, however the DSK development kit is not. If the group’s product were ever to go to market, the relevant portions of the DSK need to be reverse engineered, including the microphone line in, the audio codec, and the five volt power supply.

Additionally, the five volt power supply needs to be converted to portable power, preferably a rechargeable battery. There are various different technologies for rechargeable batteries that the team has not been able to explore. Future work on this subject would entail making a design choice and then implementing it.

Finally, the final product needs a reasonable production cost. The prototype will determine whether or not it is feasible to produce a functional product within the price range of

its consumer demand. From there a decision can be made whether or not this design is ultimately viable or not.

In conclusion, the team's project was very successful. The concept of using MFCCs to identify certain bird species based on their bird song has been proven. If completed, this product could increase the enjoyment of bird watching and even become an attachment for binoculars for more precise aim with the unidirectional microphone. The team believes that the success of this project could potentially lead to a marketable product.

Appendices

The following sections are appendices containing the full MATLAB results, MATLAB code, and C code.

Appendix A: MATLAB Test Results

This section documents the full MATLAB results and includes the code used to obtain such results. Each figure shows five different data graphs. The first signal is the time-domain signal of an “unknown” bird being passed into the system. The second signal is the time-domain signal of a bird from the database. The third signal shows eight MFCCs from the unknown bird, with each color corresponding with a different MFCC. The fourth signal shows the same eight MFCCs from the known bird from the database. Finally, the fifth graph correlates the MFCCs from the unknown bird with the MFCCs from the known bird.

```
>> mfcc_auto('CapeMayWarbler (10).wav',8,16000)

third_fit =
    0

second_fit =
CapeMayWarbler (1).wav

max =
   0.8655

x =
      NaN    0.9099    0.9208    0.9134    0.8876    0.9184    0.5902    0.9180

ans =
CapeMayWarbler (2).wav
```

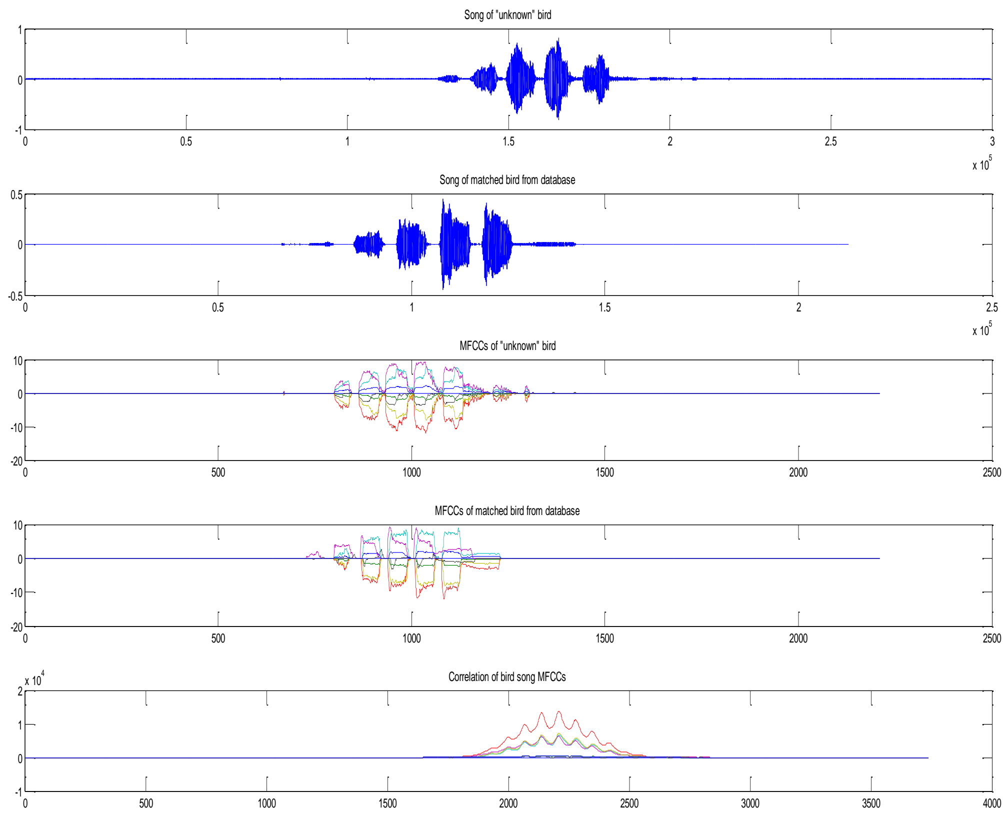

Figure 21: Cape May Warbler song comparison

```
>> mfcc_auto('CapeMayWarbler (3).wav',8,16000)

third_fit =
    0

second_fit =
    0

max =
   0.8326

x =
      NaN    0.8405    0.8768    0.8388    0.8429    0.8363    0.7646    0.8286

ans =
CapeMayWarbler (1).wav
```

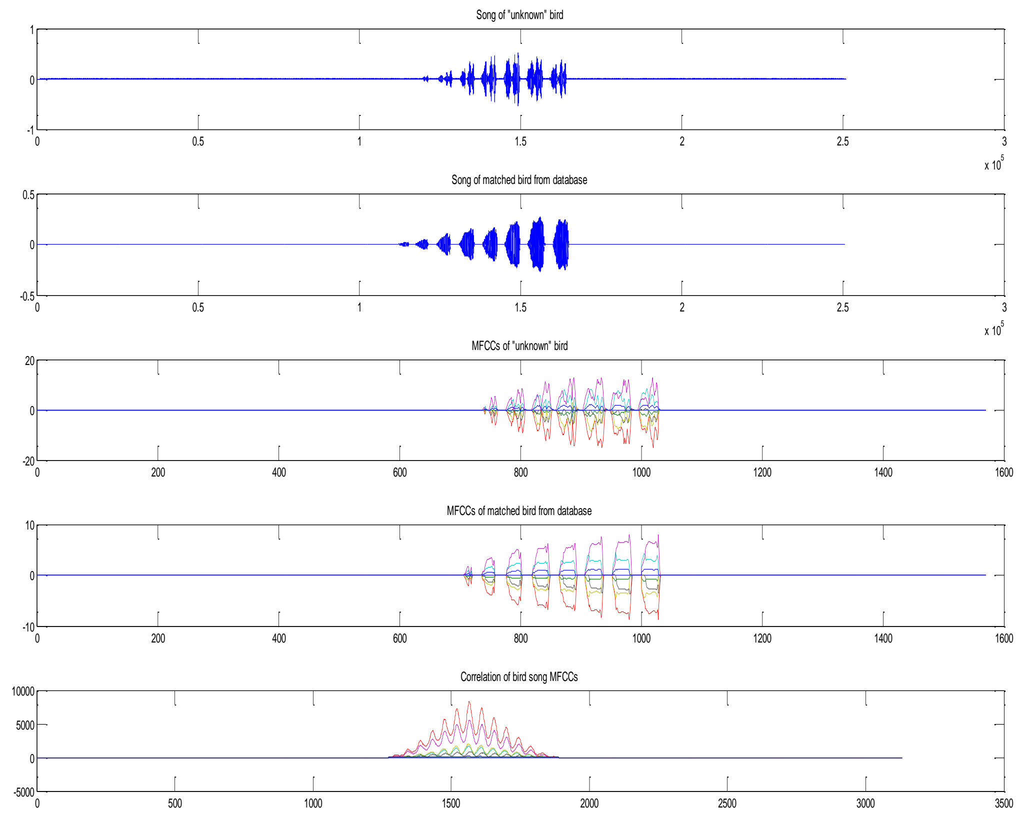

Figure 22: Cape May Warbler song comparison

```
>> mfcc_auto('CapeMayWarbler (4).wav',8,16000)

third_fit =
    0

second_fit =
    0

max =
   0.7941

x =
     NaN    0.7935    0.8451    0.8017    0.8087    0.7880    0.7180    0.8035

ans =
CapeMayWarbler (1).wav
```

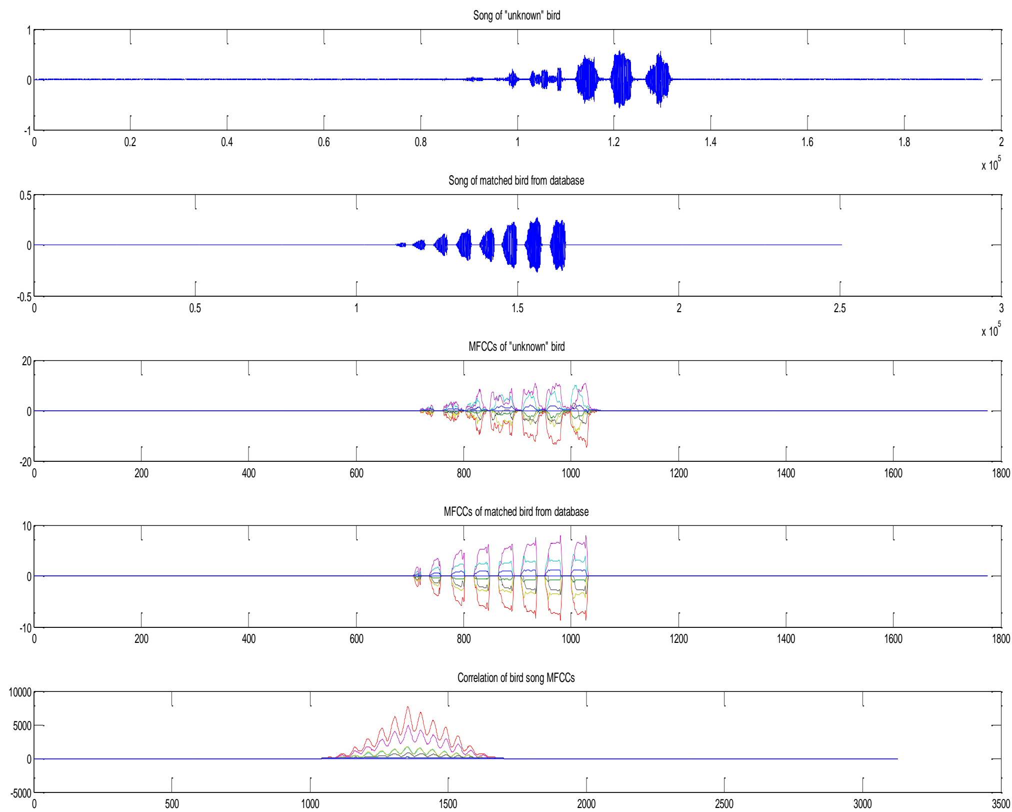

Figure 23: Cape May Warbler song comparison

```
>> mfcc_auto('CapeMayWarbler (5).wav',8,16000)

third_fit =
    0

second_fit =
CapeMayWarbler (1).wav

max =
   0.7870

x =
     NaN    0.8793    0.8733    0.8851    0.7663    0.8806    0.3855    0.8386

ans =
CapeMayWarbler (2).wav
```

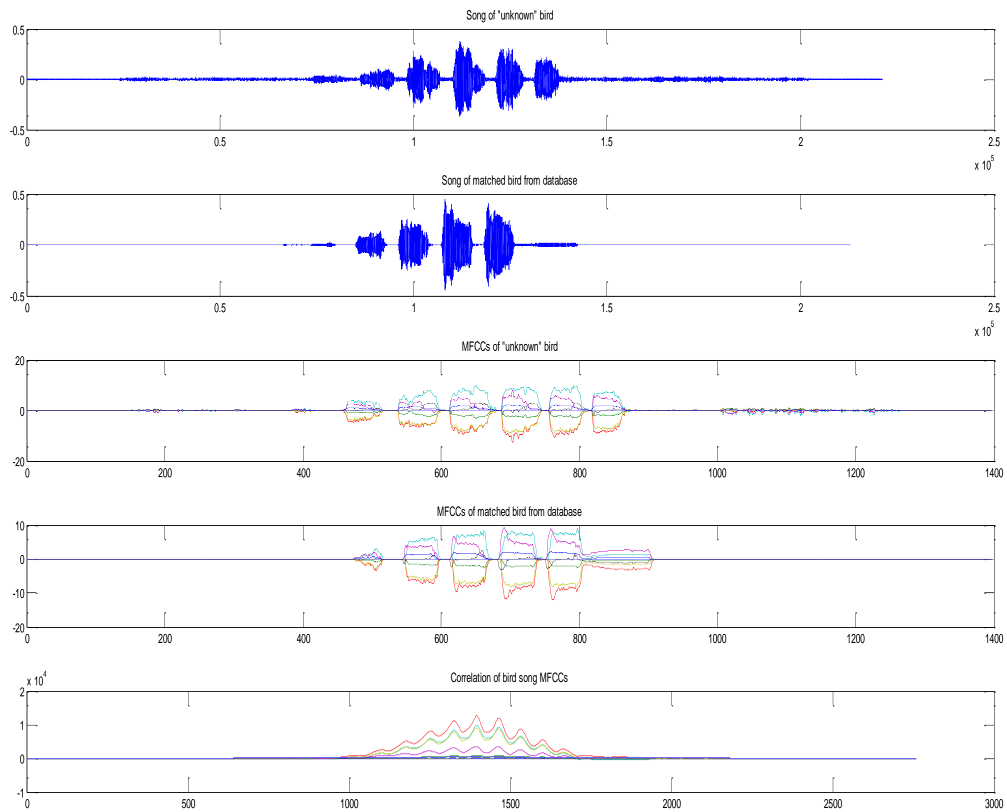

Figure 24: Cape May Warbler song comparison

```
>> mfcc_auto('CapeMayWarbler (9).wav',8,16000)

third_fit =
CapeMayWarbler (1).wav

second_fit =
CapeMayWarbler (2).wav

max =
  0.9382

x =
    NaN    0.9355    0.9542    0.9410    0.9634    0.9523    0.8544    0.9663

ans =
CapeMayWarbler (8).wav
```

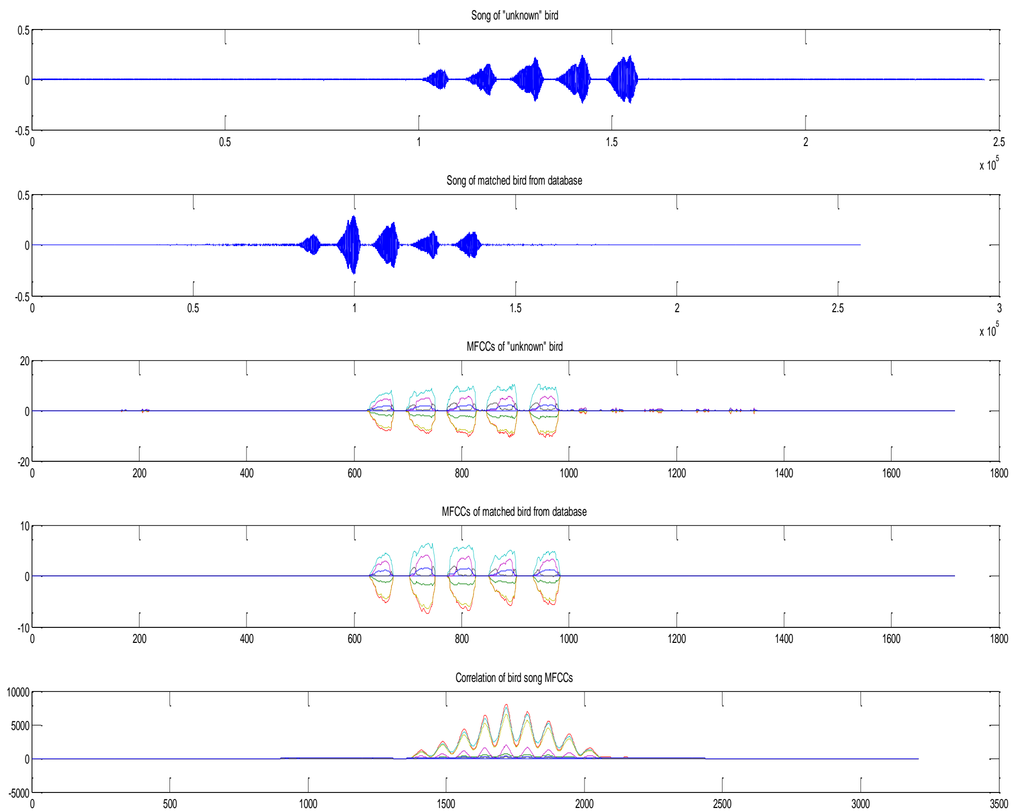

Figure 25: Cape May Warbler song comparison

```
>> mfcc_auto('MagnoliaWarbler (2).wav',8,16000)

third_fit =
    0

second_fit =
CapeMayWarbler (1).wav

max =
   0.5989

x =
      NaN    0.6562    0.6981    0.6502    0.6470    0.6484    0.4107    0.4815

ans =
MagnoliaWarbler (1).wav
```

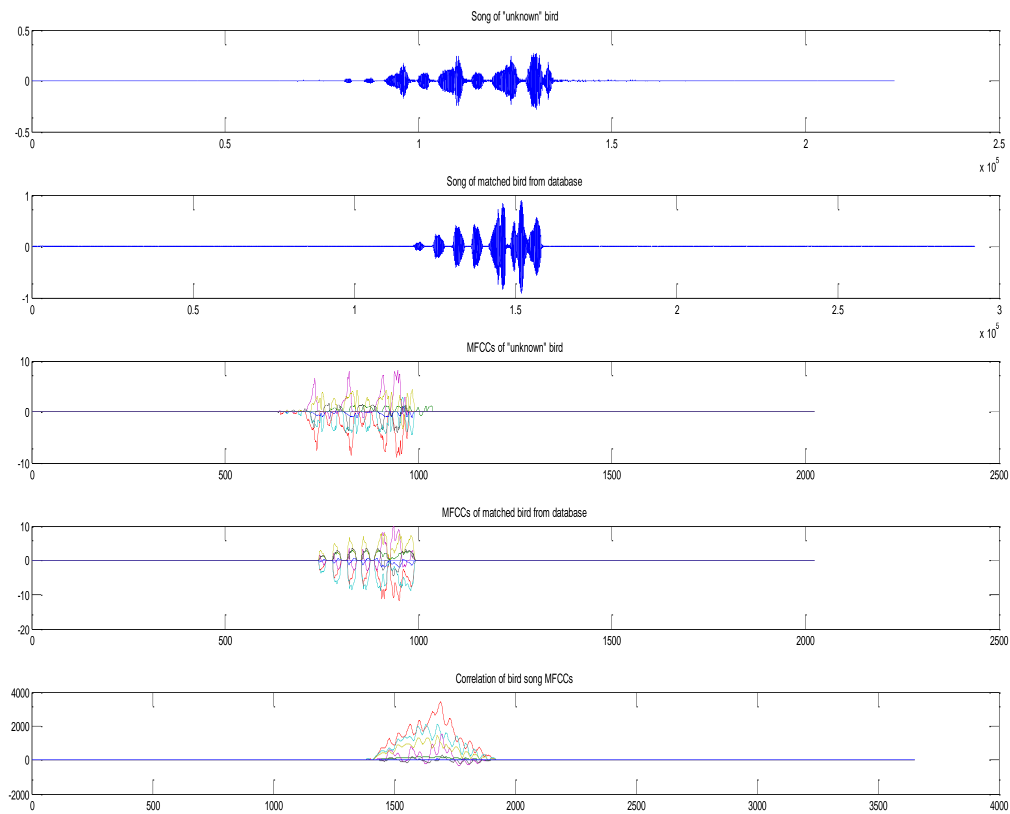

Figure 26: Magnolia Warbler song comparison

```
>> mfcc_auto('MagnoliaWarbler (4).wav',8,16000)

third_fit =
MagnoliaWarbler (1).wav

second_fit =
MagnoliaWarbler (3).wav

max =
   0.7324

x =
     NaN    0.7448    0.8344    0.7612    0.8221    0.7389    0.6369    0.5885

ans =
MagnoliaWarbler (6).wav
```

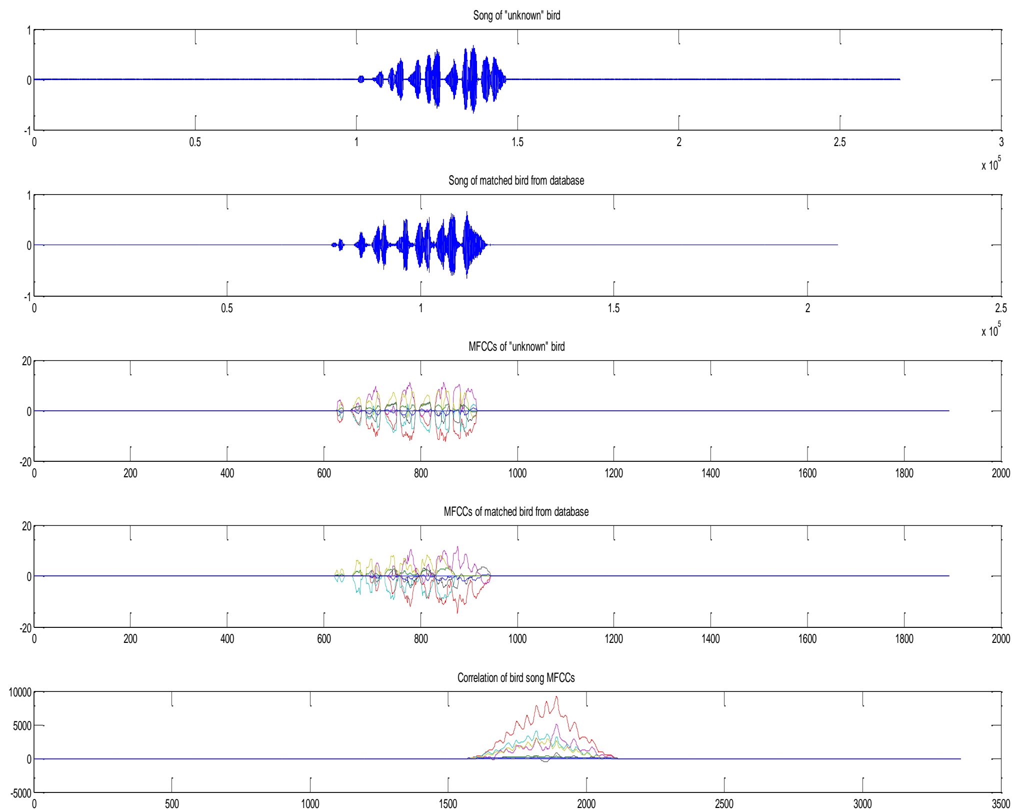

Figure 27: Magnolia Warbler song comparison

```
>> mfcc_auto('MagnoliaWarbler (5).wav',8,16000)

third_fit =
CapeMayWarbler (6).wav

second_fit =
MagnoliaWarbler (1).wav

max =
  0.7158

x =
     NaN    0.7255    0.8297    0.7995    0.6772    0.7766    0.6047    0.5973

ans =
MagnoliaWarbler (3).wav
```

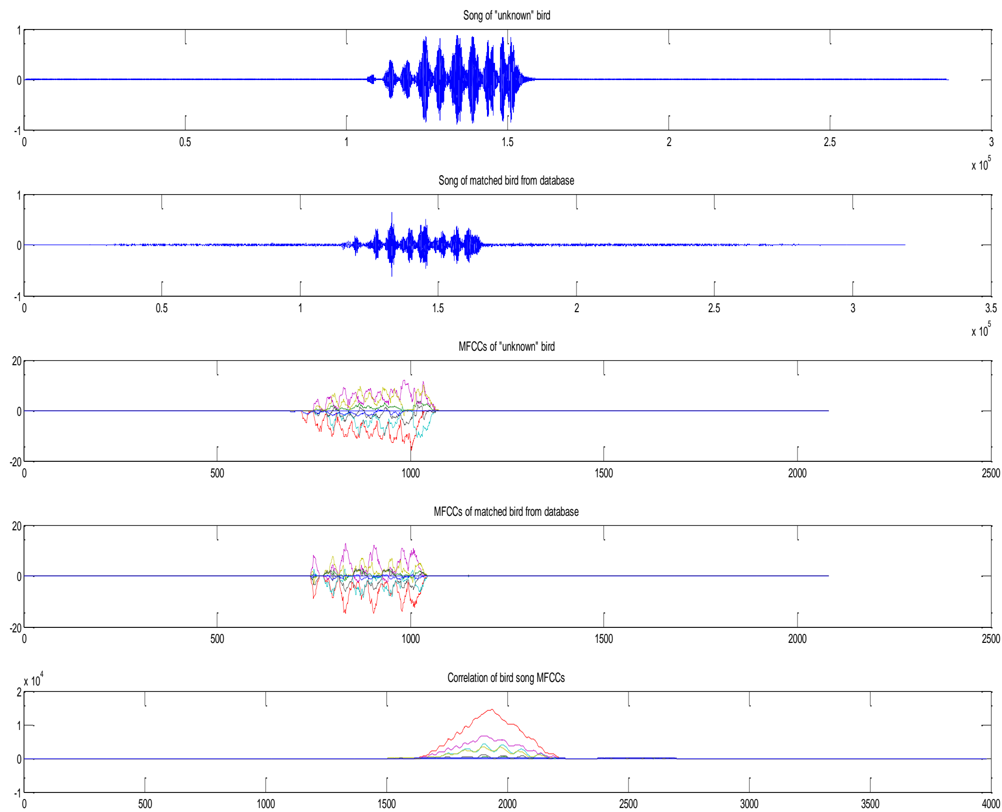

Figure 28: Magnolia Warbler song comparison

```
>> mfcc_auto('MagnoliaWarbler (8).wav',8,16000)

third_fit =
MagnoliaWarbler (3).wav

second_fit =
MagnoliaWarbler (6).wav

max =
   0.9387

x =
      NaN    0.9585    0.9530    0.9549    0.9367    0.9428    0.9145    0.9103

ans =
MagnoliaWarbler (7).wav
```

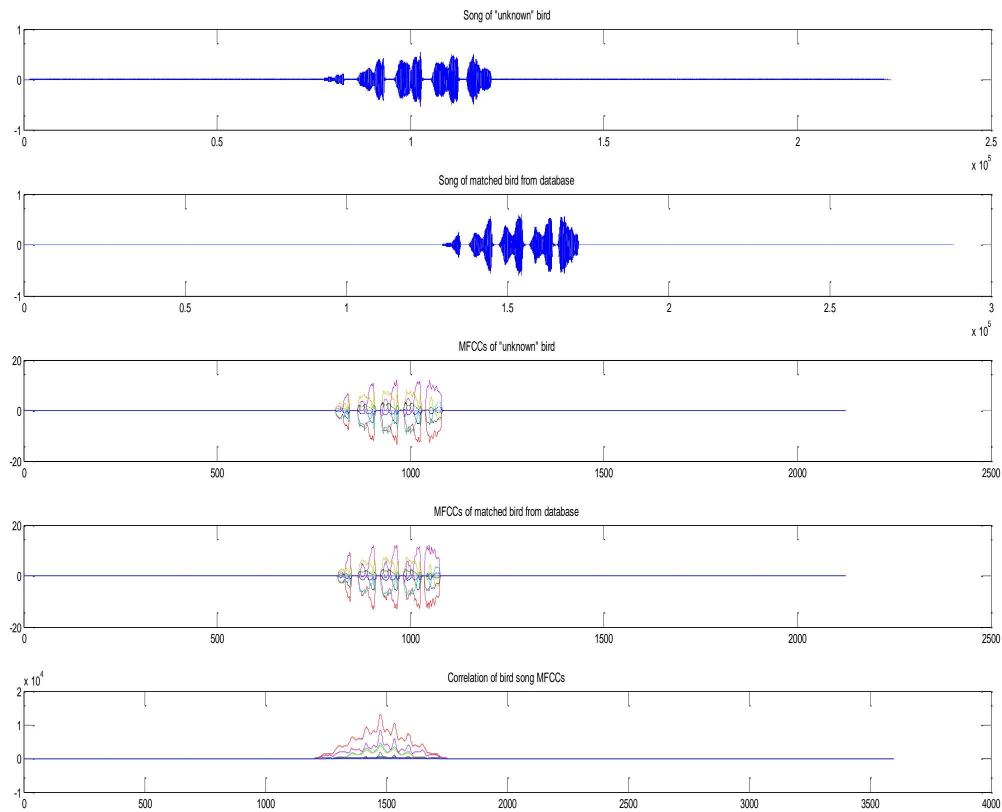

Figure 29: Magnolia Warbler song comparison

```
>> mfcc_auto('MagnoliaWarbler (9).wav',8,16000)

third_fit =
MagnoliaWarbler (3).wav

second_fit =
MagnoliaWarbler (6).wav

max =
  0.9415

x =
    NaN    0.9636    0.9484    0.9602    0.9335    0.9487    0.9167    0.9198

ans =
MagnoliaWarbler (7).wav
```

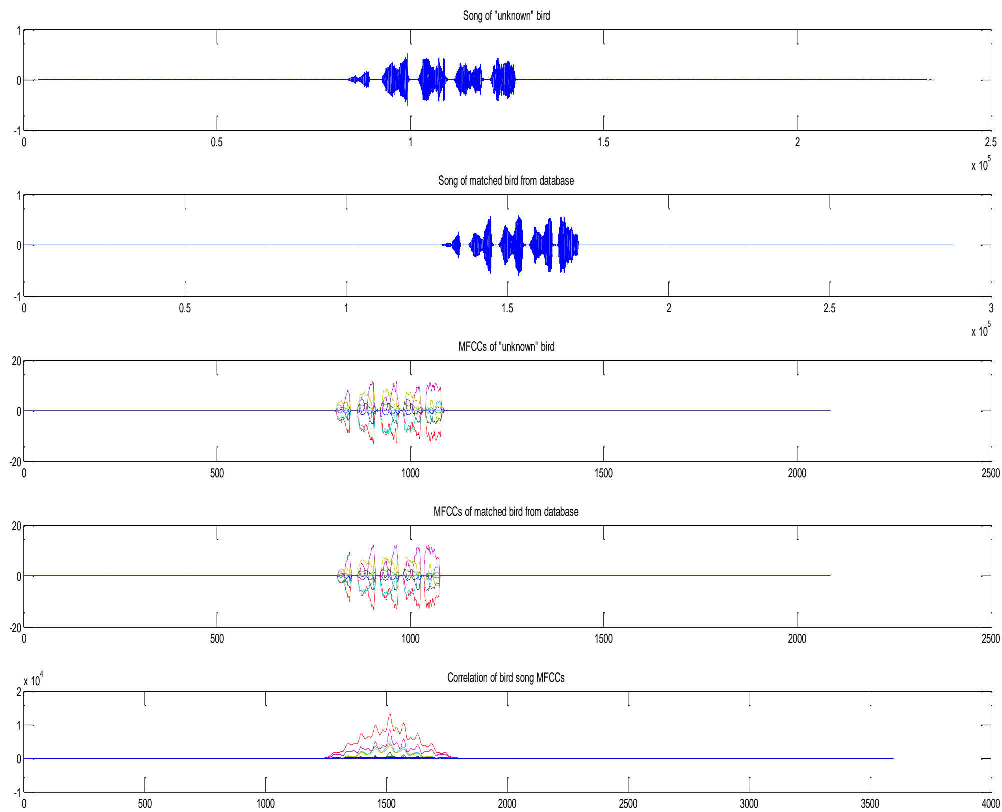

Figure 30: Magnolia Warbler song comparison

```
>> mfcc_auto('MourningWarbler (9).wav',8,16000)

third_fit =
CapeMayWarbler (6).wav

second_fit =
MagnoliaWarbler (1).wav

max =
  0.7042

x =
     NaN    0.7193    0.8145    0.7226    0.7324    0.7363    0.5642    0.6403

ans =
MourningWarbler (1).wav
```

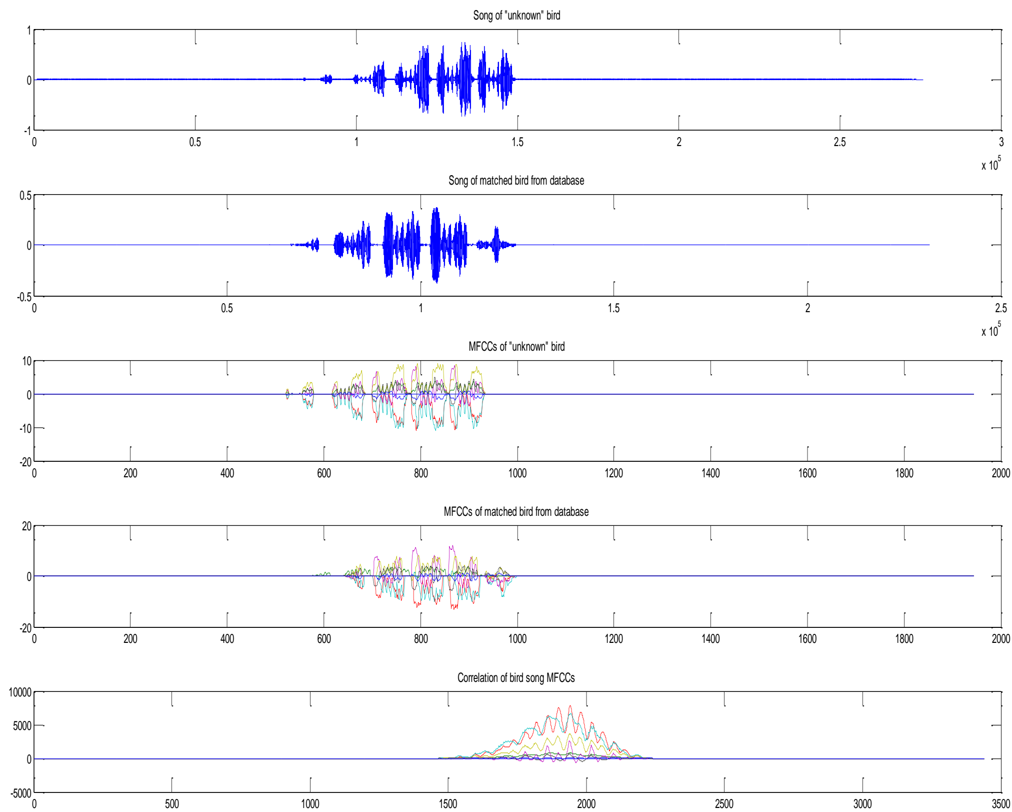

Figure 31: Mourning Warbler song comparison

```
>> mfcc_auto('MourningWarbler (2).wav',8,16000)

third_fit =
MagnoliaWarbler (1).wav

second_fit =
MagnoliaWarbler (3).wav

max =
   0.9782

x =
     NaN    0.9829    0.9839    0.9810    0.9798    0.9752    0.9728    0.9718

ans =
MourningWarbler (1).wav
```

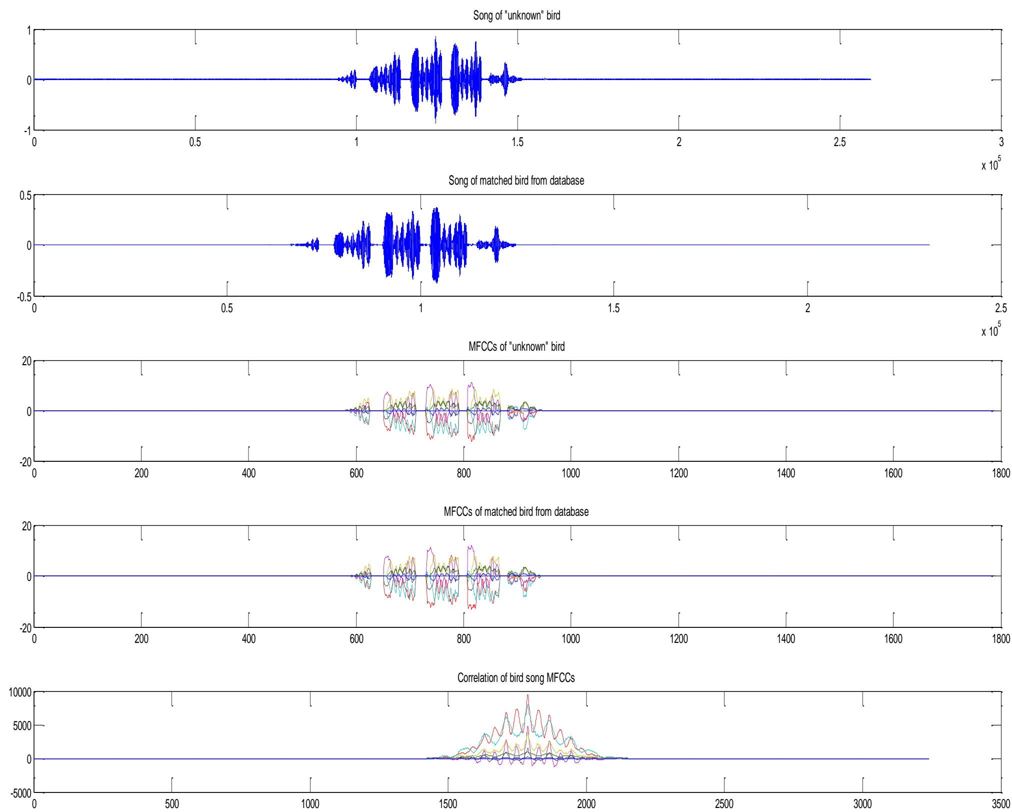

Figure 32: Mourning Warbler song comparison

```
>> mfcc_auto('MourningWarbler (8).wav',8,16000)

third_fit =
MagnoliaWarbler (1).wav

second_fit =
MagnoliaWarbler (7).wav

max =
  0.5905

x =
    NaN    0.7839    0.7393    0.7839    0.3272    0.6754    0.6094    0.2143

ans =
MagnoliaWarbler (10).wav
```

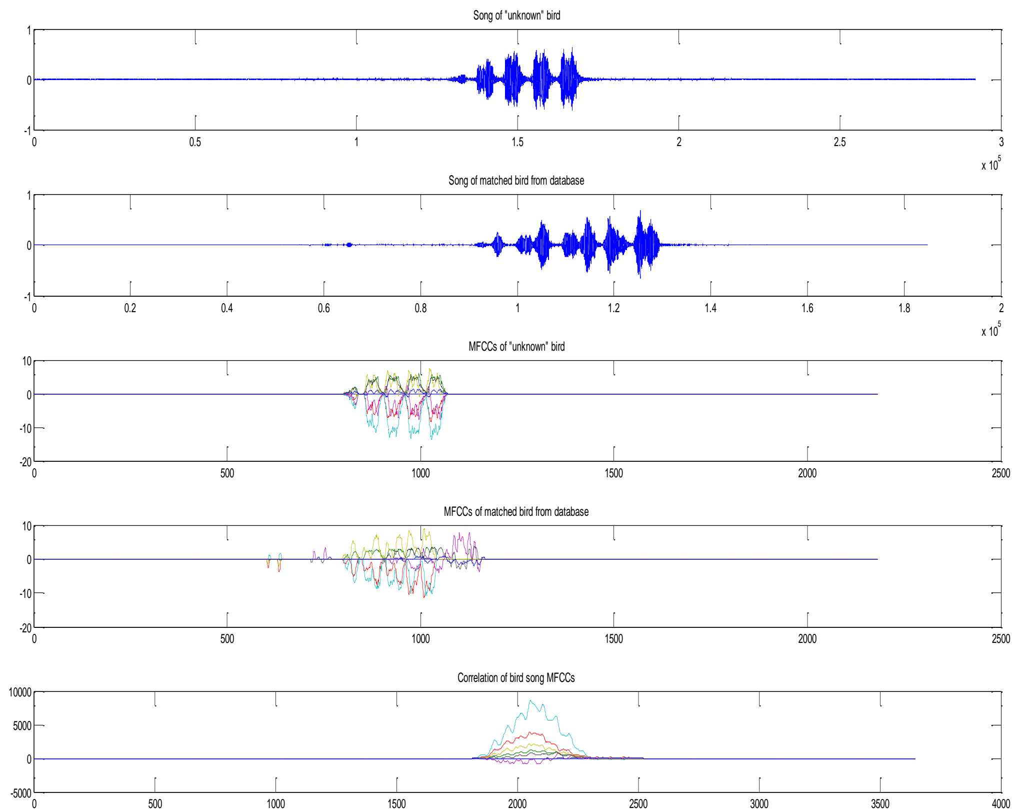

Figure 33: Mourning Warbler song comparison

```
>> mfcc_auto('MourningWarbler (10).wav',8,16000)

third_fit =
MagnoliaWarbler (1).wav

second_fit =
MagnoliaWarbler (3).wav

max =
   0.7031

x =
     NaN    0.7728    0.8500    0.7863    0.6421    0.7602    0.5750    0.5349

ans =
MourningWarbler (1).wav
```

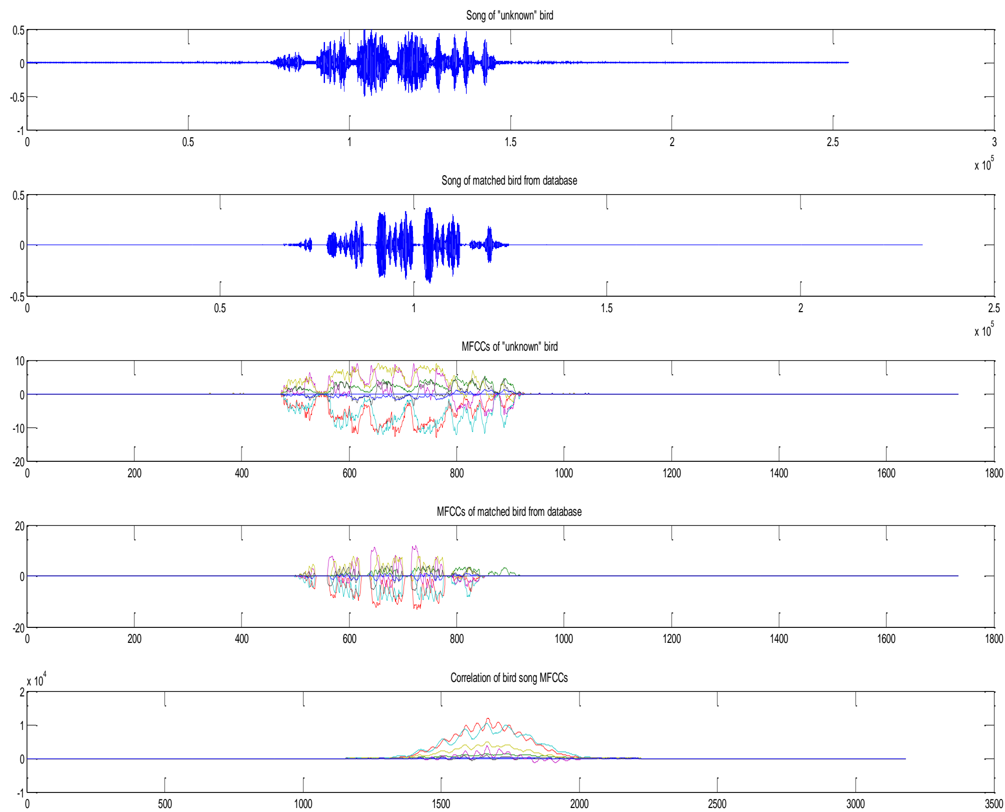

Figure 34: Mourning Warbler song comparison

```
>> mfcc_auto('CarolinaWren (7).wav',8,16000)

third_fit =
MagnoliaWarbler (10).wav

second_fit =
MourningWarbler (5).wav

max =
   0.7115

x =
      NaN    0.7560    0.7839    0.7394    0.6852    0.7611    0.6225    0.6321

ans =
CarolinaWren (1).wav
```

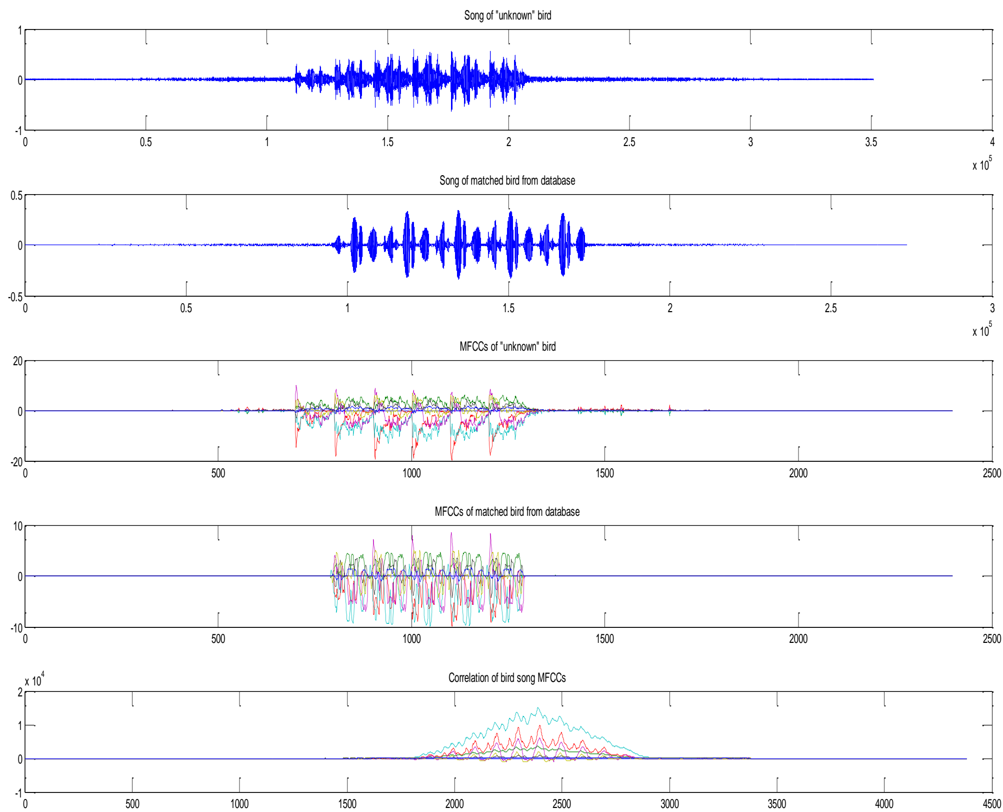

Figure 35: Carolina Wren song comparison

```
>> mfcc_auto('CarolinaWren (8).wav',8,16000)

third_fit =
MourningWarbler (5).wav

second_fit =
CarolinaWren (1).wav

max =
   0.5139

x =
      NaN    0.5698    0.4056    0.5554    0.5976    0.4777    0.4453    0.5454

ans =
CarolinaWren (5).wav
```

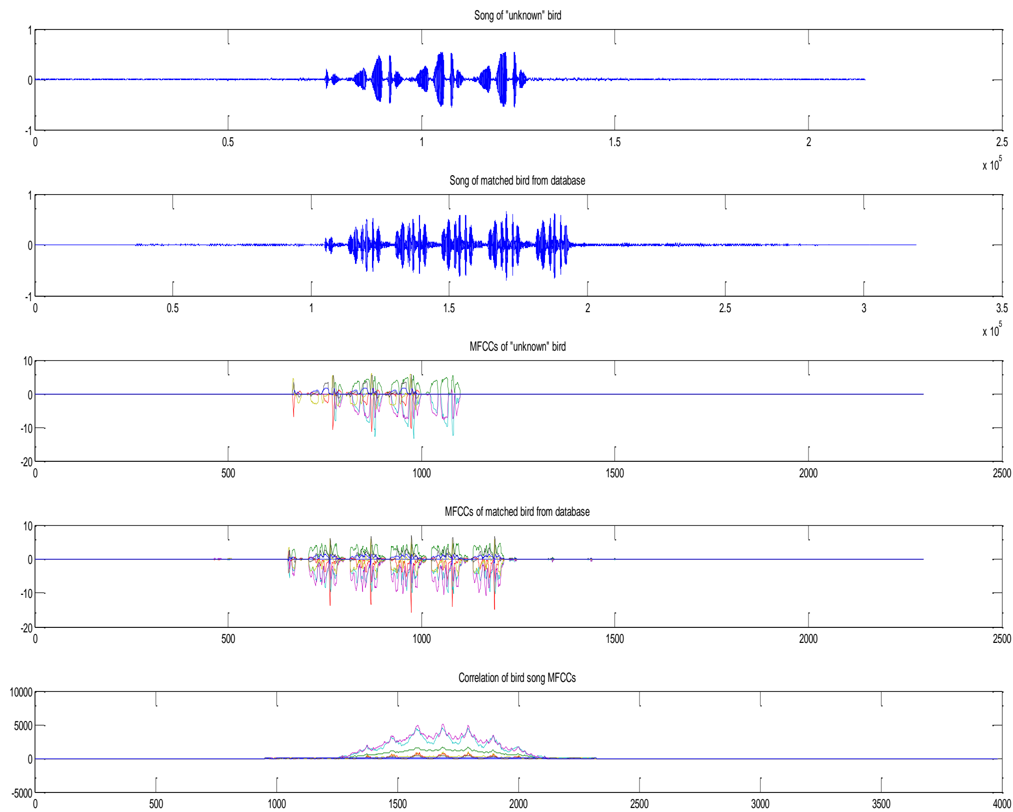

Figure 36: Carolina Wren song comparison

```
>> mfcc_auto('CarolinaWren (9).wav',8,16000)

third_fit =
MourningWarbler (6).wav

second_fit =
CarolinaWren (1).wav

max =
  0.6178

x =
    NaN    0.6659    0.6026    0.6267    0.6405    0.6885    0.5950    0.5052

ans =
CarolinaWren (2).wav
```

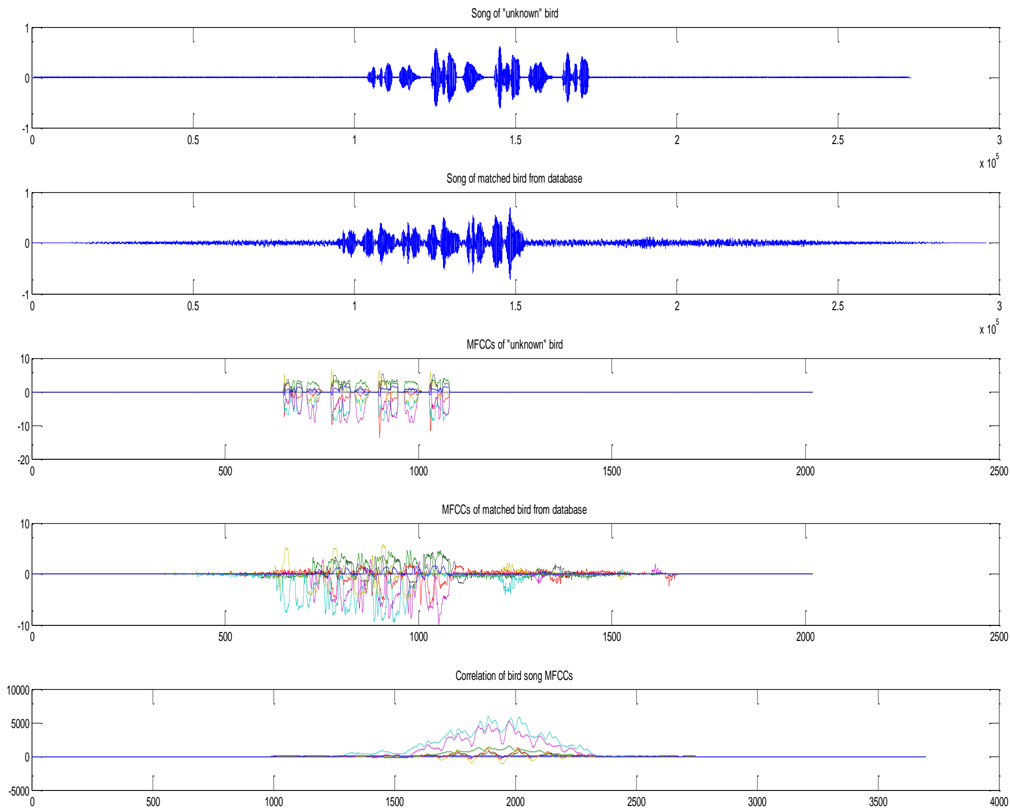

Figure 37: Carolina Wren song comparison

```
>> mfcc_auto('CarolinaWren (10).wav',8,16000)

third_fit =
MourningWarbler (5).wav

second_fit =
MourningWarbler (6).wav

max =
   0.5455

x =
     NaN    0.6534    0.5236    0.6568    0.5851    0.4905    0.4355    0.4738

ans =
CarolinaWren (2).wav
```

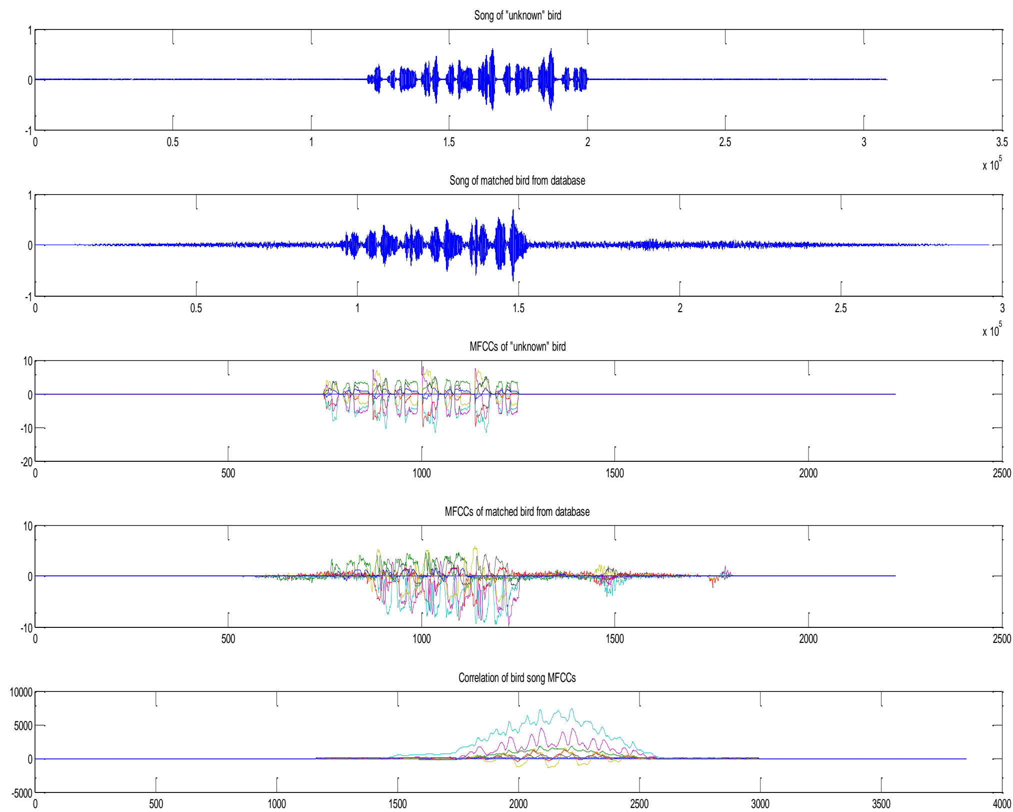

Figure 38: Carolina Wren song comparison

```
>> mfcc_auto('Carolina_Wren_Bird.wav',8,16000)

third_fit =
MourningWarbler (7).wav

second_fit =
CarolinaWren (1).wav

max =
   0.6178

x =
     NaN    0.7906    0.5959    0.7694    0.6810    0.2547    0.5677    0.6653

ans =
CarolinaWren (5).wav
```

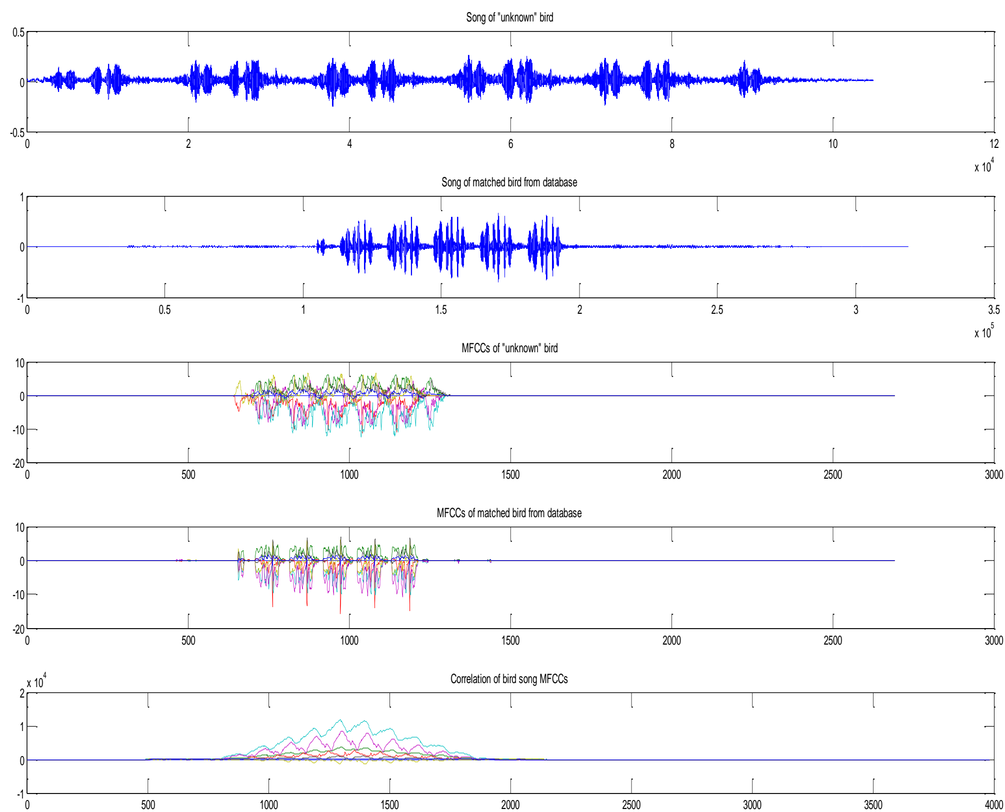

Figure 39: Carolina Wren song comparison

Appendix B: MATLAB Source Code

Bird Finder with Database

```
function best_fit = mfcc_auto( data1, num, fs )
%MFCC correlation calculation
max = 0;
second_fit = 0;
third_fit = 0;
best_fit = 0;

%Create database of bird calls
database_entry = strvcat('CapeMayWarbler (1).wav','CapeMayWarbler
(2).wav','CapeMayWarbler (6).wav','CapeMayWarbler (8).wav','MagnoliaWarbler
(1).wav','MagnoliaWarbler (3).wav','MagnoliaWarbler (6).wav','MagnoliaWarbler
(7).wav','MagnoliaWarbler (10).wav','MourningWarbler
(1).wav','MourningWarbler (3).wav','MourningWarbler (4).wav','MourningWarbler
(5).wav','MourningWarbler (6).wav','MourningWarbler (7).wav','CarolinaWren
(1).wav','CarolinaWren (2).wav','CarolinaWren (3).wav','CarolinaWren
(5).wav','CarolinaWren (6).wav');

%Find the size of the database
S = size(database_entry);

%Compare the MFCC correlation vectors of each bird call
% with the unknown bird call to find the best fit
for i = 1:S(1)
    x = mfcc_nooutput(data1,database_entry(i,:),num,fs);
    compared(i) = mean(x(2:num));
    if compared(i) > max
        max = compared(i);
        third_fit = second_fit;
        second_fit = best_fit;
        best_fit = database_entry(i,:);
    end
end

%Output the best fit
third_fit
second_fit
max
mfcc(data1,best_fit,num,fs);

end
```

MFCC Comparison Code

```
function x = mfcc_nooutput( data1, data2, num, fs )
%MFCC correlation calculation

%Read in the wave files
y1 = wavread(data1);
y2 = wavread(data2);

%Calculate MFCCs
```

```
z1 = kannumfcc(num,y1,fs);
z2 = kannumfcc(num,y2,fs);

%Find the size of the MFCCs
[s1,q] = size(z1);
[s2,q] = size(z2);

%Compare the sizes to fix any differences in size
% between the two files
if(s1 ~= s2)
    if(s1 > s2)
        z2 = padarray(z2,(s1-s2),0,'post');
    end
    if(s1 < s2)
        z1 = padarray(z1,(s2-s1),0,'post');
        s1 = s2;
    end
end

%Go through each MFCC infividually and find their
% cross correlation values
s = size(z1);
for i=2:num
    [C,I] = max(xcorr(z1(:,i),z2(:,i)));
    g(:,i) = xcorr(z1(:,i),z2(:,i));
    if(I < (s1))
        w1(1:(s+(s1-I)),i) = padarray(z1(:,i),((s1)-I),0,'pre');
        w2(1:(s+(s1-I)),i) = padarray(z2(:,i),((s1)-I),0,'post');
    end
    if(I > (s1))
        w1(1:(s+(I-s1)),i) = padarray(z1(:,i),(I-(s1)),0,'post');
        w2(1:(s+(I-s1)),i) = padarray(z2(:,i),(I-(s1)),0,'pre');
    end
    if(I == (s1))
        w1 = z1;
        w2 = z2;
    end
end

%Output the correlation vector for comparison
for i=1:num
    x(i) = corr(w1(:,i),w2(:,i));
end


end
```

Appendix C: DSP Source Code

source.c

```
//MQP

#define CHIP_6713
#define DEBUG_MODE 0

//include files
#include <stdio.h>
#include <c6x.h>
#include <csl.h>
#include <csl_mcbsp.h>
#include <csl_irq.h>

#include "dsk6713.h"
#include "dsk6713_aic23.h"
#include "fft.h"
#include "kannumfcc1.2.h"
#include "corr.h"

//Uint32 fs=DSK6713_AIC23_FREQ_16KHZ;
Uint32 fs=DSK6713_AIC23_FREQ_44KHZ;

//#define REC_LEN 163840      //10 seconds of recording at 16384 Hz
//#define REC_LEN 81920       //5 seconds of recording at 16384 Hz
#define REC_LEN 220500        //5 seconds of recording at 44100 Hz

int var;
short inout;
float buffer[REC_LEN];
#pragma DATA_SECTION(buffer,".EXT_RAM");        //write buffer to external memory

//Filtering Prototypes and Variables

//High Pass Filter
const int BL_High = 11;
const float B_High[11] = {
  -0.03768008947, -0.05519120023, -0.08344765007,  -0.1095543131,  -0.1280400008,
    0.8652742505,  -0.1280400008,  -0.1095543131, -0.08344765007, -0.05519120023,
  -0.03768008947
};

//Low Pass Filter
const int BL_Low = 31;
const float B_Low[31] = {
  0.004959571641,  0.02198479138,-0.007192821242, -0.01577280462,0.0003676032065,
   0.02300152369, 0.007212243043, -0.02984294109,  -0.0206382703,  0.03626932949,
   0.04415991902, -0.04150832444,  -0.0936364457,  0.04492767528,    0.314013809,
```

```
    0.4538732469,    0.314013809,  0.04492767528,  -0.0936364457, -0.04150832444,
    0.04415991902,  0.03626932949,  -0.0206382703, -0.02984294109, 0.007212243043,
    0.02300152369,0.0003676032065, -0.01577280462,-0.007192821242,  0.02198479138,
   0.004959571641
};

//variables for filtering
float samples_high[11];
float samples_low[31];
float result;

//variables for MFCCs
float fmatrix[numFrames][NUM_COEFFS];
float fmatrix2000[2000][NUM_COEFFS];

//kannumfcc
float num = NUM_COEFFS;
int i, j, k;
//int a = 1;
//float b[] = {1, -0.97};                //a and b are high pass filter coefficients
float frame[N];
COMPLEX F[n]; //FFT needs to be a power of 2 to work
float spectr1[fn];
float spectr[fn];
float c[fn];
float melf[24][257];
float Ce1 = 0, Ce2 = 0, Ce = 0;
float coeffs[NUM_COEFFS];
float hamming[N];
COMPLEX *input;
float noFrames; //maximum no of frames in speech sample
float lifter[NUM_COEFFS];
int len = REC_LEN;
float mean;
float max = 0;
float cutoff;

#pragma DATA_ALIGN(F,sizeof(COMPLEX));             //align F for fft()

//Variables for correlation
float c1d[length];
float d1d[length];
float maxr = 0;
float avgr = 0;
float corrs[CORR_LEN];
float corrs2d[NUM_COEFFS];
float r;

//Variables for search algorithm
int bird = 0;
float maxcorr = 0;
```

```
//Bird Call FMatrix Library Variables
#pragma DATA_SECTION(capemay1,".EXT_RAM");
#pragma DATA_SECTION(carolina1, ".EXT_RAM");
#pragma DATA_SECTION(magnolia1, ".EXT_RAM");
#pragma DATA_SECTION(mourning1, ".EXT_RAM");
//#pragma DATA_SECTION(capemay2,".EXT_RAM");
//#pragma DATA_SECTION(capemay1cutoff, ".EXT_RAM");
//#pragma DATA_SECTION(carolina1cutoff, ".EXT_RAM");
void main(){

	//initialize Board
	comm_poll();
	//initialize DIPs
	DSK6713_DIP_init();
	//initialize LEDs
	DSK6713_LED_init();

	if(DEBUG_MODE == 1){
		printf("\n\n*********PROGRAM STARTED IN DEBUG MODE*********\n\n");
	} else {
		printf("\n\n*********PROGRAM STARTED*********\n\n");
	}

	while(1){

		//initialize filtering arrays to zero
		for(i = 0; i < BL_High; i++){
			samples_high[i] = 0;

		}
		for(i = 0; i < BL_Low; i++){
			samples_low[i] = 0;

		}

		while(1){

			if(DSK6713_DIP_get(3) == 0){
				//turn LED3 on to indicate recording in progress
				printf("Recording in progress...\n");

				DSK6713_LED_on(3);
				//filter samples as they come in
				for(i = 0; i < REC_LEN; i++){

					inout = input_sample();
					buffer[i] = inout;

				}
				//turn LED3 off when buffer full
```

```
				DSK6713_LED_off(3);

				printf("...finished\n");

				break;
			}
		};
		var = 0;

		//Low Pass Filter - Stop 10kHz+
		if(DEBUG_MODE == 1){
			printf("Low pass filter in progress...");
		}
		for(i = 0; i < REC_LEN; i++){

			result = 0;

			//Update array samples
			for(j = BL_Low-2; j >= 0; j--){
				//move all samples down one
				samples_low[j+1] = samples_low[j];
			}

			//add new sample
			samples_low[0] = buffer[i];

			//Filter
			for(j = 0; j < BL_Low; j++){
				result += (samples_low[j] * B_Low[j]);
			}

			buffer[i] = result;

		}
		if(DEBUG_MODE == 1){
			printf("...finished\n");
		}

		//End Low Pass Filter

		//*********End Filtering********

		//Insert MFCC Code here - LED on and off when complete
		DSK6713_LED_on(1);
		DSK6713_LED_on(2);

		//allocate fmatrix to zero
		for(i=0;i<numFrames;i++){
			for(j=0;j<NUM_COEFFS;j++){
				fmatrix[i][j]=0;
			}
```

```
}

//kannumfcc(buffer, REC_LEN, fmatrix);

//****************kannumfcc.c************************
noFrames = floor(len/FrameStep);          //maximum no of frames in speech sample
//lifter vector index
for(i = 0; i < NUM_COEFFS; i++){
        lifter[i] = i+1;
        lifter[i] = 1 + floor(num/2) * (sin(lifter[i]*PI/num));
} //raised sine lifter version


//normalizes to compensate for mic vol differences
//not sure we have to do this
/*
for(i = 0; i < len; i++){
        mean += abs(buffer[i]);
        if(max < buffer[i]){
                max = buffer[i];
        }
}
mean = mean / len;
if(mean > 0.01){
        for(i = 0; i < len; i++){
                buffer[i] = buffer[i]/max;
        }
}
*/
//end normlization


//generate hamming window
create_hamming(hamming, N);

//create mel filter bank
create_mel_bank(melf);

printf("MFCC Calculations in progress...\n");

for(i = 1; i <= noFrames-2; i++){
        if(DEBUG_MODE == 1){

                if(i == 50){
                        printf("computing 50th frame\n");
                }
                if(i == 250){
                        printf("computing 250th frame\n");
                }
        }
        Ce1 = 0;
```

```
Ce2 = 0;
Ce = 0;
for(j = 1; j <= N; j++){
	frame[j-1] = buffer[(i-1)*FrameStep+(j-1)];			//holds
individual frames
	//frame energy
	Ce1 += (frame[j-1] * frame[j-1]);
	if(Ce1 < (2*pow(10, -22))){
		Ce2 = (2*pow(10, -22));
	} else {
		Ce2 = Ce1;
	}
}
Ce = log(Ce2);

if(DEBUG_MODE == 1){
	if(i == 1){
		printf("calculated frame energy\n");
	}
}

//multiply each frame with hamming window
for(j = 0; j < n; j++){
	F[j].re = hamming[j] * frame[j];
	F[j].im = 0;
}

if(DEBUG_MODE == 1){
	if(i == 1){
		printf("multplied hamming window\n");
	}
}

//compute FFT
//our FFT does not take inputs, fft will be rewritten in F
//initialize complex FFT input array with known values
input = &F[0];

fft(input, n);

if(DEBUG_MODE == 1){
	if(i == 1){
		printf("performed FFT\n");
	}
}

//result is mel-scale filtered, up to halfn of F
for(k = 0; k < fn; k++){
	spectr1[k] = 0;
	for(j = 1; j <= halfn; j++){
```

```
					spectr1[k] += melf[k][j-1]*(pow(imag_mag(F[j-1].re, F[j-
1].im),2));
				}
				spectr1[k] = log10(spectr1[k]);
				if(spectr1[k] < pow(10, -22)){
					spectr[k] = pow(10, -22);
				} else {
					spectr[k] = spectr1[k];
				}
			}

			if(DEBUG_MODE == 1){
				if(i == 1){
					printf("results are mel-scale filtered\n");
				}
			}

			//Compute DCT
			dct(spectr, c, fn);

			if(DEBUG_MODE == 1){
				if(i == 1){
					printf("performed dct\n");
				}
			}

			//obtains DCT, changes to cepstral domain
			c[0] = Ce;

			for(j = 0; j < NUM_COEFFS; j++){
				coeffs[j] = c[j];

				fmatrix[i][j] = coeffs[j] * lifter[j];
			}

			if(DEBUG_MODE == 1){
				if(i == 1){
					printf("wrote to fmatrix\n");
					printf("...first frame finished\n");
				}
			}

		}

		//************Correlation****************

		printf("Correlation started...\n");


		//CapeMay1
		maxr = 0;
```

```
avgr = 0;

for(i = 0; i < NUM_COEFFS; i++){

	//pad fmatrix array with zeros to length 2000 to match other fmatrices

	for(j = 0; j < length; j++){
		/*
		if(j < numFrames){
			fmatrix2000[j][i] = fmatrix[j][i];
		} else {
			fmatrix2000[j][i] = 0.01;
		}

		if(capemay1[j][i] != 0){
			c1d[j] = fmatrix[j][i];
			d1d[j] = capemay1[j][i];
		} else {
			c1d[j] = fmatrix[j][i];
			d1d[j] = 0.01;
		}
		*/
		c1d[j] = fmatrix[j][i];
		d1d[j] = capemay1[j][i];

	}

	r = cross_corr_wrap(c1d, d1d, corrs, NUM_COEFFS);
	corrs2d[i] = r;
	if(r > maxr){
		maxr = r;
	}
}
//Do not count first correlation
for(i = 1; i < NUM_COEFFS; i++){
	avgr += corrs2d[i];
}
avgr /= NUM_COEFFS-1;

maxcorr = avgr; //set avg correlation to current maximum
bird = 1;		//set current bird to cape may 1
if(DEBUG_MODE == 1){
	printf("Max corr capemay1: %f\n", maxr);
}
printf("Avg corr capemay1: %f\n", avgr);

if(DEBUG_MODE == 1){
	for(i = 0; i < NUM_COEFFS; i++){
		printf("%f\n", corrs2d[i]);
	}
}
```

```
//carolina1
maxr = 0;
avgr = 0;

for(i = 0; i < NUM_COEFFS; i++){

	//pad fmatrix array with zeros to length 2000 to match other fmatrices

	for(j = 0; j < length; j++){

		c1d[j] = fmatrix[j][i];
		d1d[j] = carolina1[j][i];
	}

	r = cross_corr_wrap(c1d, d1d, corrs, NUM_COEFFS);
	corrs2d[i] = r;
	if(r > maxr){
		maxr = r;
	}
}
//Do not count first correlation
for(i = 1; i < NUM_COEFFS; i++){
	avgr += corrs2d[i];
}
avgr /= NUM_COEFFS-1;

//if correlation is highest, set maxcorr and highest bird type
if(avgr > maxcorr){
	maxcorr = avgr;
	bird = 2;
}

if(DEBUG_MODE == 1){
	printf("Max corr carolina1: %f\n", maxr);
}
printf("Avg corr carolina1: %f\n", avgr);

if(DEBUG_MODE == 1){
	for(i = 0; i < NUM_COEFFS; i++){
		printf("%f\n", corrs2d[i]);
	}
}

//Magnolia1
maxr = 0;
avgr = 0;

for(i = 0; i < NUM_COEFFS; i++){

	for(j = 0; j < length; j++){
```

```
			c1d[j] = fmatrix[j][i];
			d1d[j] = magnolia1[j][i];
		}

		r = cross_corr_wrap(c1d, d1d, corrs, NUM_COEFFS);
		corrs2d[i] = r;
		if(r > maxr){
			maxr = r;
		}
	}
	//Do not count first correlation
	for(i = 1; i < NUM_COEFFS; i++){
		avgr += corrs2d[i];
	}
	avgr /= NUM_COEFFS-1;

	//if correlation is highest, set maxcorr and highest bird type
	if(avgr > maxcorr){
		maxcorr = avgr;
		bird = 3;
	}

	if(DEBUG_MODE == 1){
		printf("Max corr magnolia1: %f\n", maxr);
	}
	printf("Avg corr magnolia1: %f\n", avgr);

	if(DEBUG_MODE == 1){
		for(i = 0; i < NUM_COEFFS; i++){
			printf("%f\n", corrs2d[i]);
		}
	}

	//Mourning1
	maxr = 0;
	avgr = 0;

	for(i = 0; i < NUM_COEFFS; i++){

		for(j = 0; j < length; j++){
			c1d[j] = fmatrix[j][i];
			d1d[j] = mourning1[j][i];
		}

		r = cross_corr_wrap(c1d, d1d, corrs, NUM_COEFFS);
		corrs2d[i] = r;
		if(r > maxr){
			maxr = r;
		}
	}
	//Do not count first correlation
```

```
for(i = 1; i < NUM_COEFFS; i++){
	avgr += corrs2d[i];
}
avgr /= NUM_COEFFS-1;

//if correlation is highest, set maxcorr and highest bird type
if(avgr > maxcorr){
	maxcorr = avgr;
	bird = 4;
}

if(DEBUG_MODE == 1){
	printf("Max corr mourning1: %f\n", maxr);
}
printf("Avg corr mourning1: %f\n", avgr);

if(DEBUG_MODE == 1){
	for(i = 0; i < NUM_COEFFS; i++){
		printf("%f\n", corrs2d[i]);
	}
}

//Print bird with highest correlation
//if highest correlation is under .50, change to no match
if(maxcorr < 0.50){
	bird = 0;
	maxcorr = 0;
}

printf("Highest Correlation:\n");
switch(bird){
	case 0:
		printf("No Match\n");
		printf("Correlation: %f\n", maxcorr);
		break;
	case 1:
		printf("Cape May Warbler1\n");
		printf("Correlation: %f\n", maxcorr);
		break;
	case 2:
		printf("Carolina Wren1\n");
		printf("Correlation: %f\n", maxcorr);
		break;
	case 3:
		printf("Magnolia Warbler1\n");
		printf("Correlation: %f\n", maxcorr);
		break;
	case 4:
		printf("Mourning Warbler1\n");
		printf("Correlation: %f\n", maxcorr);
		break;
```

```
				default:
					printf("Default Case Reached\n");
					break;
			}


			//turn off LEDs to indicate signal processing complete
			DSK6713_LED_off(1);
			DSK6713_LED_off(2);

			//***********Play back section****************
			while(1){
				if((DSK6713_DIP_get(0) == 0) /*&& (var == 0)*/){
					//turn LED0 to indicate play back
					DSK6713_LED_on(0);

					printf("Playback in progress...\n");

					for(i = 0; i<REC_LEN; i++){
						//play back
						inout = buffer[i];
						output_sample(inout*10);
					}
					//var = 1;
					//turn LED0 off when play back is complete
					DSK6713_LED_off(0);

					printf("...finished\n");

					break;

				}


			};

		}


}
```

kannumfcc1.2.c

```
//Kannumfcc1.2.c

//The whole project.....
#include "kannumfcc1.2.h"

void kannumfcc(COMPLEX sample[], int len, float fmatrix[500][8]){
	float num = NUM_COEFFS;
	int i, j, k;
```

```
//int a = 1;
//float b[] = {1, -0.97};              //a and b are high pass filter coefficients
COMPLEX frame[N], F[N];
float spectr1[fn];
float spectr[fn];
float c[fn];
float melf[24][257];
float Ce1 = 0, Ce2 = 0, Ce = 0;
float coeffs[NUM_COEFFS], ncoeffs[NUM_COEFFS];
float hamming[N];
COMPLEX *input;

float noFrames = floor(len/FrameStep);  //maximum no of frames in speech sample
float lifter[NUM_COEFFS];                                          //lifter vector index
for(i = 0; i < NUM_COEFFS; i++){
        lifter[i] = i+1;
        lifter[i] = 1 + floor(num/2) * (sin(lifter[i]*PI/num));
} //raised sine lifter version


//normalizes to compensate for mic vol differences
//not sure we have to do this
/*
float mean;
float max = 0;
for(i = 0; i < len; i++){
        mean += abs(sample[i]);
        if(max < sample[i]){
                max = sample[i];
        }
}
mean = mean / len;
if(mean > 0.01;){
        for(i = 0; i < len; i++){
                sample[i] = sample[i]/max;
        }
}
*/
//end normlization

//segment the signal into overlapping frames and compute MFCC coefficients

for(i = 0; i < halfn; i++){
        spectr1[i] = 0;
        spectr[i] = 0;
}


//generate hamming window
create_hamming(hamming, N);
```

```
	//create mel filter bank
	create_mel_bank(melf);


	for(i = 1; i <= noFrames-2; i++){
		for(j = 1; j <= N; j++){
			frame[j-1].re = sample[(i-1)*FrameStep+(j-1)].re;			//holds
individual frames
			//frame energy
			//Ce1 += (frame[j-1].re * frame[j-1].re) + 2*(frame[j-1].re + frame[j-1].im) +
(frame[j-1].im * frame[j-1].im); //sqaure imaginary number???
			//Imaginary numbers = 0
			Ce1 += (frame[j-1].re * frame[j-1].re);
			if(Ce1 < (2*pow(10, -22))){
				Ce2 = (2*pow(10, -22));
			} else {
				Ce2 = Ce1;
			}
		}
		Ce = log(Ce2);

		//filter frame using a and b
		//not sure we have to do this...

		//multiply each frame with hamming window
		for(j = 0; j < N; j++){
			F[j].re *= hamming[j];
		}

		//compute FFT
		//our FFT does not take inputs, fft will be rewritten in F
		//initialize complex FFT input array with known values
		input = &F[0];

		fft(input, N);

		//result is mel-scale filtered, up to halfn of F
		for(k = 0; k < fn; k++){
			for(j = 1; j <= halfn; j++){
				spectr1[k] += log10(melf[k][j-1]*(pow(imag_mag(F[j-1].re, F[j-
1].im),2)));
			}
			if(spectr1[k] < pow(10, -22)){
				spectr[k] = pow(10, -22);
			} else {
				spectr[k] = spectr1[k];
			}
		}

		//Compute DCT
		dct(spectr, c, fn);
```

```
			//obtains DCT, changes to cepstral domain
			c[0] = Ce;

			for(j = 0; j < NUM_COEFFS; j++){
				coeffs[j] = c[j];
				ncoeffs[j] = coeffs[j] * lifter[j];
				fmatrix[i][j] = ncoeffs[j];
			}

		}

		return;
}

//creates a hamming window of length len
void create_hamming(float hamming[], int len){
		int i;
		for(i = 0; i < len; i++){
			hamming[i] = 0.54 - 0.46 * cos(2*PI * ((float)i/(len-1)));
		}
		return;
}

//finds the magnitude if an imaginary number
float imag_mag(float real, float imag){
		float ans = sqrt( pow(real, 2) + pow(imag, 2) );
		return ans;
}

//performs a DCT-II
void dct(float input[], float output[], int M){
		int k, m;
		int K = M;
		float temp = 0;
		for(k = 1; k <= K; k++){

			for(m = 1; m <= M; m++){
				temp += input[m-1] * cos((PI*(2*m-1)*(k-1))/(2*M));
			}

			if(k == 1){
				output[k-1] = (1/sqrt(M)) * temp;
			} else {
				output[k-1] = sqrt(2/(float)M) * temp;
			}

			temp = 0;
		}
		return;
}
```

fft.c

```
//fft.c

//Performs TI's optimized FFT function

#include "fft.h"

// align data (nothing works if you omit these pragma!!!!!!!!!!)
#pragma DATA_ALIGN(w,sizeof(COMPLEX))   //align w

// global variables
COMPLEX w[FFT_N/RADIX];                    // array of complex twiddle factors
float DELTA = 2.0*PI/FFT_N;
short iw[FFT_N/2],ix[FFT_N];               // indices for bit reversal

void fft(COMPLEX data[], int len)
{
	int i;

	// compute first N/2 twiddle factors
	for(i=0;i<len/RADIX;i++){
		w[i].re = cos(DELTA*i);
		w[i].im = sin(DELTA*i);              // negative imag component
		//iw[i] = 0;
		//ix[i] = 0;
		//ix[(len/RADIX)+i] = 0;
	}

	digitrev_index(iw,(float)len/RADIX,RADIX);//produces index for bitrev() W
	bitrev(w,iw,(float)len/RADIX);             //bit reverse W
	cfftr2_dit(data,w,len) ;                   //TI floating-pt complex FFT
	digitrev_index(ix, len, RADIX);     //produces index for bitrev() X
	bitrev(data,ix,len);                       //freq scrambled->bit-reverse X
}
```

corr.c

```
//MQP
//corr.c

//This function cross correlates two arrays

#include "corr.h"
#include "kannumfcc1.2.h"

float cross_corr(float x[], float y[], float corrs[], int nn){

	int i,j, k = 0;
	float mx,my,sx,sy,sxy,denom,r, maxr = 0;
	int delay;
```

```
	int maxdelay = MAXDELAY;

	/* Calculate the mean of the two series x[], y[] */
	mx = 0;
	my = 0;
	for (i=0;i<nn;i++) {
		mx += x[i];
		my += y[i];
	}
	mx /= nn;
	my /= nn;

	/* Calculate the denominator */
	sx = 0;
	sy = 0;
	for (i=0;i<nn;i++) {
		sx += (x[i] - mx) * (x[i] - mx);
		sy += (y[i] - my) * (y[i] - my);
	}
	denom = sqrt(sx*sy);

	/* Calculate the correlation series */
	for (delay=-maxdelay; delay<maxdelay; delay++) {
		sxy = 0;
		for (i=0;i<nn;i++) {
			j = i + delay;
			if (j < 0 || j >= nn)
				continue;
			else
			sxy += (x[i] - mx) * (y[j] - my);
		}
		/*
		if(denom == 0){
			r = 0;
		} else {
			r = sxy / denom;
		}
		*/
		r = sxy / denom;
		corrs[k] = r;
		k++;
		if(r > maxr){
			maxr = r;
		}

	  /* r is the correlation coefficient at "delay" */
	}
	return maxr;

}
```

```
float cross_corr_wrap(float x[], float y[], float corrs[], int nn){

	int i,j, k = 0;
	float mx,my,sx,sy,sxy,denom,r, maxr = 0;
	int delay;
	int maxdelay = MAXDELAY;

	/* Calculate the mean of the two series x[], y[] */
	mx = 0;
	my = 0;
	for (i=0;i<nn;i++) {
		mx += x[i];
		my += y[i];
	}
	mx /= nn;
	my /= nn;

	/* Calculate the denominator */
	sx = 0;
	sy = 0;
	for (i=0;i<nn;i++) {
		sx += (x[i] - mx) * (x[i] - mx);
		sy += (y[i] - my) * (y[i] - my);
	}
	denom = sqrt(sx*sy);

	/* Calculate the correlation series */
	for (delay=-maxdelay;delay<maxdelay;delay++) {
		sxy = 0;
		for (i=0;i<nn;i++) {
			j = i + delay;
			while (j < 0){
				j += nn;
			}
			j %= nn;
			sxy += (x[i] - mx) * (y[j] - my);
		}
		r = sxy / denom;

		corrs[k] = r;
		k++;
		if(r > maxr){
			maxr = r;
		}

		/* r is the correlation coefficient at "delay" */
	}
	return maxr;
}
```

mfcc_bank.c

```
//mfcc_bank.c

//This file creates a sparse matrix which contains the mel bank

#include "mfcc_bank.h"

void create_mel_bank(float bank[24][257]){
	//intialize values to zero
	int i, j;
	//i think it was 24 x 257?
	for(i = 0; i < 24; i++){
		for(j = 0; j < 257; j++){
			bank[i][j] = 0;
		}
	}

	bank	[	0	]	[	1	]	=	1.6667	;
	bank	[	0	]	[	2	]	=	0.84	;
	bank	[	1	]	[	2	]	=	1.16	;
	bank	[	1	]	[	3	]	=	1.4875	;
	bank	[	2	]	[	3	]	=	0.5125	;
	bank	[	1	]	[	4	]	=	0.2514	;
	bank	[	2	]	[	4	]	=	1.7486	;
	bank	[	2	]	[	5	]	=	1.1134	;
	bank	[	3	]	[	5	]	=	0.8866	;
	bank	[	2	]	[	6	]	=	0.0589	;
	bank	[	3	]	[	6	]	=	1.9411	;
	bank	[	3	]	[	7	]	=	1.0766	;
	bank	[	4	]	[	7	]	=	0.9234	;
	bank	[	3	]	[	8	]	=	0.1572	;
	bank	[	4	]	[	8	]	=	1.8428	;
	bank	[	4	]	[	9	]	=	1.2931	;
	bank	[	5	]	[	9	]	=	0.7069	;
	bank	[	4	]	[	10	]	=	0.4781	;
	bank	[	5	]	[	10	]	=	1.5219	;
	bank	[	5	]	[	11	]	=	1.7068	;
	bank	[	6	]	[	11	]	=	0.2932	;
	bank	[	5	]	[	12	]	=	0.9749	;
	bank	[	6	]	[	12	]	=	1.0251	;
	bank	[	5	]	[	13	]	=	0.2784	;
	bank	[	6	]	[	13	]	=	1.7216	;
	bank	[	6	]	[	14	]	=	1.6142	;
	bank	[	7	]	[	14	]	=	0.3858	;
	bank	[	6	]	[	15	]	=	0.9793	;
	bank	[	7	]	[	15	]	=	1.0207	;
	bank	[	6	]	[	16	]	=	0.3714	;
	bank	[	7	]	[	16	]	=	1.6286	;
	bank	[	7	]	[	17	]	=	1.7881	;
	bank	[	8	]	[	17	]	=	0.2119	;
```

```
bank [ 7 ] [ 18 ] = 1.2275 ;
bank [ 8 ] [ 18 ] = 0.7725 ;
bank [ 7 ] [ 19 ] = 0.6881 ;
bank [ 8 ] [ 19 ] = 1.3119 ;
bank [ 7 ] [ 20 ] = 0.1681 ;
bank [ 8 ] [ 20 ] = 1.8319 ;
bank [ 8 ] [ 21 ] = 1.6664 ;
bank [ 9 ] [ 21 ] = 0.3336 ;
bank [ 8 ] [ 22 ] = 1.1815 ;
bank [ 9 ] [ 22 ] = 0.8185 ;
bank [ 8 ] [ 23 ] = 0.7125 ;
bank [ 9 ] [ 23 ] = 1.2875 ;
bank [ 8 ] [ 24 ] = 0.2584 ;
bank [ 9 ] [ 24 ] = 1.7416 ;
bank [ 9 ] [ 25 ] = 1.8181 ;
bank [ 10 ] [ 25 ] = 0.1819 ;
bank [ 9 ] [ 26 ] = 1.391 ;
bank [ 10 ] [ 26 ] = 0.609 ;
bank [ 9 ] [ 27 ] = 0.9762 ;
bank [ 10 ] [ 27 ] = 1.0238 ;
bank [ 9 ] [ 28 ] = 0.573 ;
bank [ 10 ] [ 28 ] = 1.427 ;
bank [ 9 ] [ 29 ] = 0.1808 ;
bank [ 10 ] [ 29 ] = 1.8192 ;
bank [ 10 ] [ 30 ] = 1.7991 ;
bank [ 11 ] [ 30 ] = 0.2009 ;
bank [ 10 ] [ 31 ] = 1.4273 ;
bank [ 11 ] [ 31 ] = 0.5727 ;
bank [ 10 ] [ 32 ] = 1.0648 ;
bank [ 11 ] [ 32 ] = 0.9352 ;
bank [ 10 ] [ 33 ] = 0.7113 ;
bank [ 11 ] [ 33 ] = 1.2887 ;
bank [ 10 ] [ 34 ] = 0.3662 ;
bank [ 11 ] [ 34 ] = 1.6338 ;
bank [ 10 ] [ 35 ] = 0.0292 ;
bank [ 11 ] [ 35 ] = 1.9708 ;
bank [ 11 ] [ 36 ] = 1.7 ;
bank [ 12 ] [ 36 ] = 0.3 ;
bank [ 11 ] [ 37 ] = 1.3782 ;
bank [ 12 ] [ 37 ] = 0.6218 ;
bank [ 11 ] [ 38 ] = 1.0634 ;
bank [ 12 ] [ 38 ] = 0.9366 ;
bank [ 11 ] [ 39 ] = 0.7553 ;
bank [ 12 ] [ 39 ] = 1.2447 ;
bank [ 11 ] [ 40 ] = 0.4537 ;
bank [ 12 ] [ 40 ] = 1.5463 ;
bank [ 11 ] [ 41 ] = 0.1584 ;
bank [ 12 ] [ 41 ] = 1.8416 ;
bank [ 12 ] [ 42 ] = 1.8689 ;
bank [ 13 ] [ 42 ] = 0.1311 ;
bank [ 12 ] [ 43 ] = 1.5852 ;
```

bank [13] [43] = 0.4148 ;
bank [12] [44] = 1.307 ;
bank [13] [44] = 0.693 ;
bank [12] [45] = 1.0341 ;
bank [13] [45] = 0.9659 ;
bank [12] [46] = 0.7663 ;
bank [13] [46] = 1.2337 ;
bank [12] [47] = 0.5033 ;
bank [13] [47] = 1.4967 ;
bank [12] [48] = 0.2451 ;
bank [13] [48] = 1.7549 ;
bank [13] [49] = 1.9915 ;
bank [14] [49] = 0.0085 ;
bank [13] [50] = 1.7423 ;
bank [14] [50] = 0.2577 ;
bank [13] [51] = 1.4973 ;
bank [14] [51] = 0.5027 ;
bank [13] [52] = 1.2564 ;
bank [14] [52] = 0.7436 ;
bank [13] [53] = 1.0195 ;
bank [14] [53] = 0.9805 ;
bank [13] [54] = 0.7864 ;
bank [14] [54] = 1.2136 ;
bank [13] [55] = 0.5571 ;
bank [14] [55] = 1.4429 ;
bank [13] [56] = 0.3313 ;
bank [14] [56] = 1.6687 ;
bank [13] [57] = 0.1091 ;
bank [14] [57] = 1.8909 ;
bank [14] [58] = 1.8902 ;
bank [15] [58] = 0.1098 ;
bank [14] [59] = 1.6747 ;
bank [15] [59] = 0.3253 ;
bank [14] [60] = 1.4623 ;
bank [15] [60] = 0.5377 ;
bank [14] [61] = 1.253 ;
bank [15] [61] = 0.747 ;
bank [14] [62] = 1.0467 ;
bank [15] [62] = 0.9533 ;
bank [14] [63] = 0.8433 ;
bank [15] [63] = 1.1567 ;
bank [14] [64] = 0.6428 ;
bank [15] [64] = 1.3572 ;
bank [14] [65] = 0.445 ;
bank [15] [65] = 1.555 ;
bank [14] [66] = 0.25 ;
bank [15] [66] = 1.75 ;
bank [14] [67] = 0.0575 ;
bank [15] [67] = 1.9425 ;
bank [15] [68] = 1.8676 ;
bank [16] [68] = 0.1324 ;

```
bank [ 15 ] [ 69 ] = 1.6801 ;
bank [ 16 ] [ 69 ] = 0.3199 ;
bank [ 15 ] [ 70 ] = 1.4951 ;
bank [ 16 ] [ 70 ] = 0.5049 ;
bank [ 15 ] [ 71 ] = 1.3125 ;
bank [ 16 ] [ 71 ] = 0.6875 ;
bank [ 15 ] [ 72 ] = 1.1321 ;
bank [ 16 ] [ 72 ] = 0.8679 ;
bank [ 15 ] [ 73 ] = 0.9539 ;
bank [ 16 ] [ 73 ] = 1.0461 ;
bank [ 15 ] [ 74 ] = 0.778 ;
bank [ 16 ] [ 74 ] = 1.222 ;
bank [ 15 ] [ 75 ] = 0.6041 ;
bank [ 16 ] [ 75 ] = 1.3959 ;
bank [ 15 ] [ 76 ] = 0.4324 ;
bank [ 16 ] [ 76 ] = 1.5676 ;
bank [ 15 ] [ 77 ] = 0.2627 ;
bank [ 16 ] [ 77 ] = 1.7373 ;
bank [ 15 ] [ 78 ] = 0.0949 ;
bank [ 16 ] [ 78 ] = 1.9051 ;
bank [ 16 ] [ 79 ] = 1.9291 ;
bank [ 17 ] [ 79 ] = 0.0709 ;
bank [ 16 ] [ 80 ] = 1.7652 ;
bank [ 17 ] [ 80 ] = 0.2348 ;
bank [ 16 ] [ 81 ] = 1.6032 ;
bank [ 17 ] [ 81 ] = 0.3968 ;
bank [ 16 ] [ 82 ] = 1.4429 ;
bank [ 17 ] [ 82 ] = 0.5571 ;
bank [ 16 ] [ 83 ] = 1.2844 ;
bank [ 17 ] [ 83 ] = 0.7156 ;
bank [ 16 ] [ 84 ] = 1.1277 ;
bank [ 17 ] [ 84 ] = 0.8723 ;
bank [ 16 ] [ 85 ] = 0.9726 ;
bank [ 17 ] [ 85 ] = 1.0274 ;
bank [ 16 ] [ 86 ] = 0.8192 ;
bank [ 17 ] [ 86 ] = 1.1808 ;
bank [ 16 ] [ 87 ] = 0.6674 ;
bank [ 17 ] [ 87 ] = 1.3326 ;
bank [ 16 ] [ 88 ] = 0.5172 ;
bank [ 17 ] [ 88 ] = 1.4828 ;
bank [ 16 ] [ 89 ] = 0.3686 ;
bank [ 17 ] [ 89 ] = 1.6314 ;
bank [ 16 ] [ 90 ] = 0.2215 ;
bank [ 17 ] [ 90 ] = 1.7785 ;
bank [ 16 ] [ 91 ] = 0.0759 ;
bank [ 17 ] [ 91 ] = 1.9241 ;
bank [ 17 ] [ 92 ] = 1.9317 ;
bank [ 18 ] [ 92 ] = 0.0683 ;
bank [ 17 ] [ 93 ] = 1.789 ;
bank [ 18 ] [ 93 ] = 0.211 ;
bank [ 17 ] [ 94 ] = 1.6476 ;
```

bank [18] [94] = 0.3524 ;
bank [17] [95] = 1.5077 ;
bank [18] [95] = 0.4923 ;
bank [17] [96] = 1.3691 ;
bank [18] [96] = 0.6309 ;
bank [17] [97] = 1.2318 ;
bank [18] [97] = 0.7682 ;
bank [17] [98] = 1.0958 ;
bank [18] [98] = 0.9042 ;
bank [17] [99] = 0.9611 ;
bank [18] [99] = 1.0389 ;
bank [17] [100] = 0.8277 ;
bank [18] [100] = 1.1723 ;
bank [17] [101] = 0.6954 ;
bank [18] [101] = 1.3046 ;
bank [17] [102] = 0.5644 ;
bank [18] [102] = 1.4356 ;
bank [17] [103] = 0.4346 ;
bank [18] [103] = 1.5654 ;
bank [17] [104] = 0.3059 ;
bank [18] [104] = 1.6941 ;
bank [17] [105] = 0.1784 ;
bank [18] [105] = 1.8216 ;
bank [17] [106] = 0.052 ;
bank [18] [106] = 1.948 ;
bank [18] [107] = 1.9267 ;
bank [19] [107] = 0.0733 ;
bank [18] [108] = 1.8025 ;
bank [19] [108] = 0.1975 ;
bank [18] [109] = 1.6793 ;
bank [19] [109] = 0.3207 ;
bank [18] [110] = 1.5572 ;
bank [19] [110] = 0.4428 ;
bank [18] [111] = 1.4362 ;
bank [19] [111] = 0.5638 ;
bank [18] [112] = 1.3161 ;
bank [19] [112] = 0.6839 ;
bank [18] [113] = 1.197 ;
bank [19] [113] = 0.803 ;
bank [18] [114] = 1.0789 ;
bank [19] [114] = 0.9211 ;
bank [18] [115] = 0.9618 ;
bank [19] [115] = 1.0382 ;
bank [18] [116] = 0.8456 ;
bank [19] [116] = 1.1544 ;
bank [18] [117] = 0.7304 ;
bank [19] [117] = 1.2696 ;
bank [18] [118] = 0.6161 ;
bank [19] [118] = 1.3839 ;
bank [18] [119] = 0.5026 ;
bank [19] [119] = 1.4974 ;

bank [18] [120] = 0.3901 ;
bank [19] [120] = 1.6099 ;
bank [18] [121] = 0.2784 ;
bank [19] [121] = 1.7216 ;
bank [18] [122] = 0.1676 ;
bank [19] [122] = 1.8324 ;
bank [18] [123] = 0.0577 ;
bank [19] [123] = 1.9423 ;
bank [19] [124] = 1.9486 ;
bank [20] [124] = 0.0514 ;
bank [19] [125] = 1.8403 ;
bank [20] [125] = 0.1597 ;
bank [19] [126] = 1.7328 ;
bank [20] [126] = 0.2672 ;
bank [19] [127] = 1.6261 ;
bank [20] [127] = 0.3739 ;
bank [19] [128] = 1.5202 ;
bank [20] [128] = 0.4798 ;
bank [19] [129] = 1.4151 ;
bank [20] [129] = 0.5849 ;
bank [19] [130] = 1.3107 ;
bank [20] [130] = 0.6893 ;
bank [19] [131] = 1.2071 ;
bank [20] [131] = 0.7929 ;
bank [19] [132] = 1.1042 ;
bank [20] [132] = 0.8958 ;
bank [19] [133] = 1.0021 ;
bank [20] [133] = 0.9979 ;
bank [19] [134] = 0.9007 ;
bank [20] [134] = 1.0993 ;
bank [19] [135] = 0.8 ;
bank [20] [135] = 1.2 ;
bank [19] [136] = 0.7 ;
bank [20] [136] = 1.3 ;
bank [19] [137] = 0.6007 ;
bank [20] [137] = 1.3993 ;
bank [19] [138] = 0.5021 ;
bank [20] [138] = 1.4979 ;
bank [19] [139] = 0.4041 ;
bank [20] [139] = 1.5959 ;
bank [19] [140] = 0.3068 ;
bank [20] [140] = 1.6932 ;
bank [19] [141] = 0.2102 ;
bank [20] [141] = 1.7898 ;
bank [19] [142] = 0.1142 ;
bank [20] [142] = 1.8858 ;
bank [19] [143] = 0.0188 ;
bank [20] [143] = 1.9812 ;
bank [20] [144] = 1.9241 ;
bank [21] [144] = 0.0759 ;
bank [20] [145] = 1.83 ;

```
bank [ 21 ] [ 145 ] = 0.17 ;
bank [ 20 ] [ 146 ] = 1.7365 ;
bank [ 21 ] [ 146 ] = 0.2635 ;
bank [ 20 ] [ 147 ] = 1.6436 ;
bank [ 21 ] [ 147 ] = 0.3564 ;
bank [ 20 ] [ 148 ] = 1.5513 ;
bank [ 21 ] [ 148 ] = 0.4487 ;
bank [ 20 ] [ 149 ] = 1.4596 ;
bank [ 21 ] [ 149 ] = 0.5404 ;
bank [ 20 ] [ 150 ] = 1.3685 ;
bank [ 21 ] [ 150 ] = 0.6315 ;
bank [ 20 ] [ 151 ] = 1.278 ;
bank [ 21 ] [ 151 ] = 0.722 ;
bank [ 20 ] [ 152 ] = 1.188 ;
bank [ 21 ] [ 152 ] = 0.812 ;
bank [ 20 ] [ 153 ] = 1.0986 ;
bank [ 21 ] [ 153 ] = 0.9014 ;
bank [ 20 ] [ 154 ] = 1.0097 ;
bank [ 21 ] [ 154 ] = 0.9903 ;
bank [ 20 ] [ 155 ] = 0.9214 ;
bank [ 21 ] [ 155 ] = 1.0786 ;
bank [ 20 ] [ 156 ] = 0.8336 ;
bank [ 21 ] [ 156 ] = 1.1664 ;
bank [ 20 ] [ 157 ] = 0.7464 ;
bank [ 21 ] [ 157 ] = 1.2536 ;
bank [ 20 ] [ 158 ] = 0.6597 ;
bank [ 21 ] [ 158 ] = 1.3403 ;
bank [ 20 ] [ 159 ] = 0.5735 ;
bank [ 21 ] [ 159 ] = 1.4265 ;
bank [ 20 ] [ 160 ] = 0.4878 ;
bank [ 21 ] [ 160 ] = 1.5122 ;
bank [ 20 ] [ 161 ] = 0.4026 ;
bank [ 21 ] [ 161 ] = 1.5974 ;
bank [ 20 ] [ 162 ] = 0.3179 ;
bank [ 21 ] [ 162 ] = 1.6821 ;
bank [ 20 ] [ 163 ] = 0.2338 ;
bank [ 21 ] [ 163 ] = 1.7662 ;
bank [ 20 ] [ 164 ] = 0.1501 ;
bank [ 21 ] [ 164 ] = 1.8499 ;
bank [ 20 ] [ 165 ] = 0.0669 ;
bank [ 21 ] [ 165 ] = 1.9331 ;
bank [ 21 ] [ 166 ] = 1.9841 ;
bank [ 22 ] [ 166 ] = 0.0159 ;
bank [ 21 ] [ 167 ] = 1.9019 ;
bank [ 22 ] [ 167 ] = 0.0981 ;
bank [ 21 ] [ 168 ] = 1.8201 ;
bank [ 22 ] [ 168 ] = 0.1799 ;
bank [ 21 ] [ 169 ] = 1.7388 ;
bank [ 22 ] [ 169 ] = 0.2612 ;
bank [ 21 ] [ 170 ] = 1.6579 ;
bank [ 22 ] [ 170 ] = 0.3421 ;
```

bank [21] [171] = 1.5775 ;
bank [22] [171] = 0.4225 ;
bank [21] [172] = 1.4976 ;
bank [22] [172] = 0.5024 ;
bank [21] [173] = 1.4181 ;
bank [22] [173] = 0.5819 ;
bank [21] [174] = 1.339 ;
bank [22] [174] = 0.661 ;
bank [21] [175] = 1.2603 ;
bank [22] [175] = 0.7397 ;
bank [21] [176] = 1.1821 ;
bank [22] [176] = 0.8179 ;
bank [21] [177] = 1.1043 ;
bank [22] [177] = 0.8957 ;
bank [21] [178] = 1.027 ;
bank [22] [178] = 0.973 ;
bank [21] [179] = 0.95 ;
bank [22] [179] = 1.05 ;
bank [21] [180] = 0.8734 ;
bank [22] [180] = 1.1266 ;
bank [21] [181] = 0.7973 ;
bank [22] [181] = 1.2027 ;
bank [21] [182] = 0.7216 ;
bank [22] [182] = 1.2784 ;
bank [21] [183] = 0.6462 ;
bank [22] [183] = 1.3538 ;
bank [21] [184] = 0.5713 ;
bank [22] [184] = 1.4287 ;
bank [21] [185] = 0.4967 ;
bank [22] [185] = 1.5033 ;
bank [21] [186] = 0.4225 ;
bank [22] [186] = 1.5775 ;
bank [21] [187] = 0.3487 ;
bank [22] [187] = 1.6513 ;
bank [21] [188] = 0.2753 ;
bank [22] [188] = 1.7247 ;
bank [21] [189] = 0.2023 ;
bank [22] [189] = 1.7977 ;
bank [21] [190] = 0.1296 ;
bank [22] [190] = 1.8704 ;
bank [21] [191] = 0.0573 ;
bank [22] [191] = 1.9427 ;
bank [22] [192] = 1.9853 ;
bank [23] [192] = 0.0147 ;
bank [22] [193] = 1.9137 ;
bank [23] [193] = 0.0863 ;
bank [22] [194] = 1.8425 ;
bank [23] [194] = 0.1575 ;
bank [22] [195] = 1.7716 ;
bank [23] [195] = 0.2284 ;
bank [22] [196] = 1.7011 ;

```
bank [ 23 ] [ 196 ] = 0.2989 ;
bank [ 22 ] [ 197 ] = 1.6309 ;
bank [ 23 ] [ 197 ] = 0.3691 ;
bank [ 22 ] [ 198 ] = 1.5611 ;
bank [ 23 ] [ 198 ] = 0.4389 ;
bank [ 22 ] [ 199 ] = 1.4915 ;
bank [ 23 ] [ 199 ] = 0.5085 ;
bank [ 22 ] [ 200 ] = 1.4224 ;
bank [ 23 ] [ 200 ] = 0.5776 ;
bank [ 22 ] [ 201 ] = 1.3535 ;
bank [ 23 ] [ 201 ] = 0.6465 ;
bank [ 22 ] [ 202 ] = 1.285 ;
bank [ 23 ] [ 202 ] = 0.715 ;
bank [ 22 ] [ 203 ] = 1.2168 ;
bank [ 23 ] [ 203 ] = 0.7832 ;
bank [ 22 ] [ 204 ] = 1.1489 ;
bank [ 23 ] [ 204 ] = 0.8511 ;
bank [ 22 ] [ 205 ] = 1.0814 ;
bank [ 23 ] [ 205 ] = 0.9186 ;
bank [ 22 ] [ 206 ] = 1.0142 ;
bank [ 23 ] [ 206 ] = 0.9858 ;
bank [ 22 ] [ 207 ] = 0.9472 ;
bank [ 23 ] [ 207 ] = 1.0528 ;
bank [ 22 ] [ 208 ] = 0.8806 ;
bank [ 23 ] [ 208 ] = 1.1194 ;
bank [ 22 ] [ 209 ] = 0.8143 ;
bank [ 23 ] [ 209 ] = 1.1857 ;
bank [ 22 ] [ 210 ] = 0.7483 ;
bank [ 23 ] [ 210 ] = 1.2517 ;
bank [ 22 ] [ 211 ] = 0.6826 ;
bank [ 23 ] [ 211 ] = 1.3174 ;
bank [ 22 ] [ 212 ] = 0.6172 ;
bank [ 23 ] [ 212 ] = 1.3828 ;
bank [ 22 ] [ 213 ] = 0.5521 ;
bank [ 23 ] [ 213 ] = 1.4479 ;
bank [ 22 ] [ 214 ] = 0.4873 ;
bank [ 23 ] [ 214 ] = 1.5127 ;
bank [ 22 ] [ 215 ] = 0.4228 ;
bank [ 23 ] [ 215 ] = 1.5772 ;
bank [ 22 ] [ 216 ] = 0.3586 ;
bank [ 23 ] [ 216 ] = 1.6414 ;
bank [ 22 ] [ 217 ] = 0.2947 ;
bank [ 23 ] [ 217 ] = 1.7053 ;
bank [ 22 ] [ 218 ] = 0.231 ;
bank [ 23 ] [ 218 ] = 1.769 ;
bank [ 22 ] [ 219 ] = 0.1676 ;
bank [ 23 ] [ 219 ] = 1.8324 ;
bank [ 22 ] [ 220 ] = 0.1045 ;
bank [ 23 ] [ 220 ] = 1.8955 ;
bank [ 22 ] [ 221 ] = 0.0417 ;
bank [ 23 ] [ 221 ] = 1.9583 ;
```

```
bank [ 23 ] [ 222 ] = 1.9792 ;
bank [ 23 ] [ 223 ] = 1.9169 ;
bank [ 23 ] [ 224 ] = 1.8549 ;
bank [ 23 ] [ 225 ] = 1.7931 ;
bank [ 23 ] [ 226 ] = 1.7317 ;
bank [ 23 ] [ 227 ] = 1.6704 ;
bank [ 23 ] [ 228 ] = 1.6095 ;
bank [ 23 ] [ 229 ] = 1.5488 ;
bank [ 23 ] [ 230 ] = 1.4883 ;
bank [ 23 ] [ 231 ] = 1.4282 ;
bank [ 23 ] [ 232 ] = 1.3682 ;
bank [ 23 ] [ 233 ] = 1.3085 ;
bank [ 23 ] [ 234 ] = 1.2491 ;
bank [ 23 ] [ 235 ] = 1.1899 ;
bank [ 23 ] [ 236 ] = 1.1309 ;
bank [ 23 ] [ 237 ] = 1.0722 ;
bank [ 23 ] [ 238 ] = 1.0138 ;
bank [ 23 ] [ 239 ] = 0.9555 ;
bank [ 23 ] [ 240 ] = 0.8975 ;
bank [ 23 ] [ 241 ] = 0.8397 ;
bank [ 23 ] [ 242 ] = 0.7822 ;
bank [ 23 ] [ 243 ] = 0.7249 ;
bank [ 23 ] [ 244 ] = 0.6678 ;
bank [ 23 ] [ 245 ] = 0.611 ;
bank [ 23 ] [ 246 ] = 0.5543 ;
bank [ 23 ] [ 247 ] = 0.4979 ;
bank [ 23 ] [ 248 ] = 0.4417 ;
bank [ 23 ] [ 249 ] = 0.3858 ;
bank [ 23 ] [ 250 ] = 0.33 ;
bank [ 23 ] [ 251 ] = 0.2745 ;
bank [ 23 ] [ 252 ] = 0.2192 ;
bank [ 23 ] [ 253 ] = 0.1641 ;
bank [ 23 ] [ 254 ] = 0.1092 ;

}
```

References

Brookes, Mike. (2009). Description of melbankm from Voicebox: Speech Processing ToolBox for MATLAB. Retrieved April 20, 2010, from http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/doc/voicebox/melbankm.html

Byers, Bruce E.. (1995). Song Types, Repertoires and Song Variability in a Population of Chestnut-Sided Warblers. The Condor, Vol. 97, No. 2, pp. 390-401

Cai et al. (2007). "Sensor network for the monitoring of ecosystem: Bird species recognition," in ISSNIP 2007: Proceedings of the 3rd International Conference on Intelligent Sensors, Sensor Networks and Information Processing, pp. 293–298.

Chassaing, Rulph. (2005). Digital signal processing and applications with the c6713 and c6416 dsk. John Wiley and Sons.

Emlen, Stephen T. (1972). An Experimental Analysis of the Parameters of Bird Song Eliciting Species Recognition. Behaviour, Vol. 41, No. 1/2, pp. 130-171

Konishi, M. (1970). Evolution of design features in the coding of species-specificity. American Zoology, Vol. 10, p. 67-72

Lee et al. (2006). Automatic recognition of animal vocalizations using averaged MFCC and linear discriminant analysis. Pattern Recognition Letters, Vol. 27, pp. 93-101.

Lee et al. (2006). Automatic recognition of bird songs using cepstral coefficients, Journal of Information Technology and Applications, Vol. 1, No. 1, May 2006, pp. 17-23

Nelson, Douglas A. (1989). The Importance of Invariant and Distinctive Features in Species Recognition of Bird Song. The Condor, Vol. 91, No. 1, pp. 120-130

Omogbenigun, Olutope Foluso. (2009). File details from MATLAB Central. Retrieved April 20, 2010, from http://www.mathworks.com/matlabcentral/fileexchange/23119-mfcc

Proakis, John & Manolakis, Dimitris. (2005). Digital signal processing. Prentice Hall.

Sakata et al. (2008). Social Modulation of Sequence and Syllable Variability in Adult Birdsong. J Neurophysiology, Vol. 99, pp. 1700-1711

Terasawa et al. (2005). Perceptual distance in timbre space. In International Conference on Auditory Display, pages 61–68

Compare Birding Apps. iBird’s comparison of birding apps on the market. Retrieved April 20, 2010, from http://ibird.com/Compare.aspx

Cross-correlation. Wikipedia. Retrieved October 13, 2010, from http://en.wikipedia.org/wiki/Cross_correlation

Discrete Fourier Transform. Wikipedia. Retrieved October 13, 2010, from http://en.wikipedia.org/wiki/Discrete_Fourier_transform

eNature, eNature.com. Retrieved October 24, 2009, from http://www.enature.com/

Identiflyer. For the Bird Inc. Retrieved October 24, 2009, from http://www.identiflyer.com/

Mel Scale. Wikipedia. Retrieved April 20, 2010, from http://en.wikipedia.org/wiki/Mel_scale

Mel-frequency Cepstrum. Wikipedia. Retrieved April 20, 2010, from http://en.wikipedia.org/wiki/Mel-frequency_cepstrum

Song Sleuth. Wildlife Acoustics. Retrieved October 25, 2009, from http://www.wildlifeacoustics.com/news/aba.html

Song Meter 2. Wildlife Acoustics. Retrieved October 25, 2009, from http://www.wildlifeacoustics.com/sm2/

System Requirements – Release 2010a. Mathworks. Retrieved November 4, 2009, from http://www.mathworks.com/support/sysreq/current_release/index.html

TMS320C6713 DSP Starter Kit (DSK). Texas Instruments. Retrieved December 2, 2009, from http://focus.ti.com/docs/toolsw/folders/print/tmdsdsk6713.html