Laboratory Assignments for Teaching Introductory Signal Processing Concepts

by

Katherine Kem

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Electrical Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2017

Author . . . .
Department of Electrical Engineering and Computer Science
Aug 18, 2017

Certified by . . . .
Adam Hartz
Lecturer
Thesis Supervisor

Accepted by . . . .
Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Laboratory Assignments for Teaching Introductory Signal Processing Concepts

by

Katherine Kem

Submitted to the Department of Electrical Engineering and Computer Science
on Aug 18, 2017, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Electrical Science and Engineering

Abstract

This thesis proposes labs for a new, applications-based signal processing class. These labs span topics in audio, image, and video processing and will combine signal processing techniques with computational tools. The goal of these labs is to improve student understanding of signal processing concepts and show them the power of signal processing in everyday applications.

Thesis Supervisor: Adam Hartz
Title: Lecturer

Acknowledgments

I have many fabulous people to thank for helping me through this process.

Adam Hartz, my thesis supervisor, for always being encouraging, helpful and not afraid to give good critical feedback. My academic advisor, Jeffrey Shapiro, for helping me navigate my academic path and providing me with delicious sea food. Braden Knight, for "linguistic and emotional support" and musical motivation. Neerja Aggarwal for being my 6-1 comrade-in-arms. The copy-editing-team of Kathleen Zhou, Kenny Friedman, Colin McDonnell and JNH. The denizens of Jinder Neutral Housing for being awesome, supportive roommates: Manjinder Singh, Dirk Beck, Laura Jarin-Lipschitz, and Matthew Tancik. David Hu for kindly letting me use his awesome photos. And finally, my incredible parents, Martha and John Kem, for supporting me throughout my entire life and letting me cry to them on the phone.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis proposes labs for an application-based class in signal processing at MIT. These labs will facilitate learning and understanding of important signal processing concepts through real-world design problems.

1.1 First Principles of Instruction

M. David Merrill, author of the book *First Principles of Instruction* [5] has spent years studying education theory to determine which education strategies and approaches are the most successful. He has identified five key First Principles as the foundations behind s successful education curriculum. To qualify as a First Principle, it "had to promote more effective, efficient, or engaging learning. When the principle had been subjected to careful study, it had to be supported by the research." These five First Principles are:

- *Problem-Centered:* Learning is promoted when learners acquire skill in the context of real world problems.
- *Activation:* Learning is promoted when learners activate existing knowledge and skill as a foundation for new skills.
- *Demonstration:* Learning is promoted when learners observe a demonstration of the skill to be learned.

- *Application:* Learning is promoted when learners apply their newly acquired skill to solve problems.

- *Integration:* Learning is promoted when learners reflect on, discuss, and defend their newly acquired skill.

According to Merrill, "it is assumed that [...] these principles are necessary for effective, efficient, and engaging instruction. If this assumption is true, then it is hypothesized that when a given instructional program or practice fails to appropriately implement one or more of these First Principles, there will be a decrement in learning and performance [5]." Ideally, classes will use a combination of all five of these principles to maximize learning and retention.

1.2 Signals at MIT

A solid understanding of signals and signal processing can be applied to nearly every area of electrical engineering, from optics and digital electronics to analog circuits and robotics. It is one of the most versatile tools in the core electrical engineering curriculum.

The following list shows the general progression of signals and signal processing classes at MIT. Some classes in this progression already have an applications based component. The course descriptions have been taken from the MIT course catalog [6]:

1. *Introductory Level*: 6.01, 6.02, 6.03, or 6.S08. These are all introductory classes at MIT that cover a range of topics. 6.02 is the only introductory subject to cover signal processing. According to the course description, the "signals module includes modeling physical channels and noise, signal design, filtering and detection, modulation, and frequency-division multiplexing." This class has a lab component.

2. *Foundations Level*: 6.003. "Presents the fundamentals of signal and system analysis. Topics include discrete-time and continuous-time signals, Fourier se-

ries and transforms, Laplace and Z transforms, and analysis of linear, time-invariant systems. Applications drawn broadly from engineering and physics, including audio and image processing, communications, and automatic control." This class has no lab component.

3. *Header Level*: 6.011. "Covers signals, systems and inference in communication, control and signal processing. Topics include input-output and state-space models of linear systems driven by deterministic and random signals; time- and transform-domain representations in discrete and continuous time; and group delay. State feedback and observers. Probabilistic models; stochastic processes, correlation functions, power spectra, spectral factorization. Least-mean square error estimation; Wiener filtering. Hypothesis testing; detection; matched filters." This class has no lab component.

4. *Graduate Level*: There are several classes at the graduate signal processing level. Some of these include a laboratory or project component.

Generally, an electrical engineering student at MIT will take this progression of signals classes, but undergraduates often stop this progression at the header-level. At this point in the progression, a student can graduate having never analyzed a physical signal within an explicit signal processing context. Some other classes touch on topics and techniques tangentially related to signal processing: systems and controls are covered in the AUS course 6.302, and transforms are used as a method of analysis in circuit classes such as 6.002 and 6.012. However, within the signals class progression itself, more advanced signals concepts are not applied to real world scenarios until the graduate level.

This thesis proposes applications-based labs for a foundation-level signal processing class. There is currently on foundation-level signals and systems class, 6.003. 6.003 uses a combination of lectures, recitations, problem sets, and exams. The lectures contain demonstrations, fulfilling the *Demonstration* principle (see Section 1.1). In having students use what they've learned to solve problems in problem sets and exams, the class uses the *Application* principle. The class progression builds on past skills to teach new topics, which touches on the *Activation* principle. However, it fails to address the *Problem-Centered* principle by not including a real world context for students in which to apply their skills. It also fails to address the *Integration* principle by having no opportunity for students to discuss their takeaways. One way to address these missing principles is to add a laboratory component to the class.

1.3 Foundations Level Class Comparison

Some of the other electrical engineering foundations-level classes already contain applications-based projects. I've compared the student feedback of these classes with the current version of 6.003.

The four classes satisfying the foundation-level requirements for the electrical engineering curriculum are:

- 6.002: Circuits and Electronics
- 6.003: Signals and Systems
- 6.004: Computation Structures
- 6.007: Electromagnetic Energy

Of these four classes, 6.003 is the only class with no laboratory or applications component. The HKN Underground Guide [3] publishes ratings for overall class score, which come from student surveys at the end of the class. The overall scores for the nine semesters most recently published (spanning from Spring 2012 to Spring 2016) have been averaged together, weighted by the number of student responses per semester, and are displayed in Table 1.1. 6.003 has the lowest overall score by a half point or more with a score of 5.1. This low score, when combined with the student comments addressed below, suggests a deficiency in the current version of the class.

In addition to this numerical feedback on classes, the HKN underground guide also collects comments from students, publishing student opinion summaries as well as direct quotes. It aggregates the feedback and publishes what students commonly

Class	Overall Score
6.002	5.6
6.003	5.1
6.004	6.1
6.007	5.6

Table 1.1: HKN Underground Guide Class Ratings

These ratings are on a scale of 1 to 7, where 1 corresponds to "very poor" and 7 corresponds to "excellent." 6.007 has a much smaller class size than the others and therefore is more susceptible to small sample bias.

found "cool" or "uncool" about the class. I've looked at the feedback for all nine semesters published since the Spring 2012 term for all four foundation-level classes. For 6.003, seven of these nine semester summaries listed the in-class demos as one of the "cool" aspects of the class. This suggests that students enjoyed seeing signals applied to real world situations. Five semester summaries said that the abstract, overly mathematical aspect of the class was "uncool." A student in the Spring 2012 class said, "this class is so mathematical sometimes, it's hard to see the relevance to real life." The Spring 2014 summary stated that "as the material was very theoretical many students felt that having more examples and applications would have helped solidify concepts."

In contrast to the highly theoretical 6.003, 6.004 has a substantial laboratory component and the highest overall score of 6.1. The summary for Spring 2016 described 6.004 as having a "balanced mix of theoretical and hands on experience" and that "students enjoyed the labs and found them useful in solidifying concepts used in lecture." Feedback consistently confirmed that the labs "helped the students gain a better understanding of the material." In addition to facilitating student learning, the labs for 6.004 made students feel accomplished and excited. A student from the Spring 2013 class said that one lab "was probably the coolest thing I have ever done...you will shed tears of joy." This feeling of pride and excitement about the assignments and material is missing from the 6.003 feedback.

This 6.004 feedback suggests that a lab component improves student understanding and motivation. However, not all labs are created equal. According to the Underground Guide class summary, students enjoyed the 6.002 labs and thought that "completing [them] helped them gain a better understanding of the material" (Spring 2014). However, one lab, in which students construct a miniature music player, had much better reception than the "long and boring" amplifier characterization lab (Spring 2013). Applying the concepts learned in class to making a real life circuit in the music player lab was more exciting for students than the more tedious amplier lab.

The labs in 6.007 received a mixed reception. Many students found the labs "interesting, reasonable, and helpful" (Spring 2015). However, some considered the labs "fun and interesting though not necessarily contributing to learning" (Fall 2014). Students did appreciate the interesting applications of the concepts learned in 6.007. A Fall 2012 student said, "[there's] always immediate applications to what you learn often in technology you use everyday" [3].

Labs should be carefully created to be fun and exciting while reinforcing concepts learned in class and facilitating understanding. According to David Merrill, "the most motivating of all events is when people realize that they can solve a problem or perform a task that they couldn't do before. Learning is the most motivating of all activities when learners can observe their progress. The key exclamation is, *'Watch me! See what I can do!'*" [5]. An applications-based laboratory component allows students to see their progress in a real world context and have an exciting result they can show off.

1.4 Goals of This Thesis

Adam Hartz and Dennis Freeman are currently working on the curriculum for a new sophomore level, applications-based signal processing class likely to be offered in Spring 2018. This year I worked with Adam Hartz in designing four of the labs and projects that the students will complete for this class. The course spans signal processing applications in image, audio, and video processing.

The currents signals class hits three of the five principles of instruction (Section

1.1): *Demonstration*, *Application*, and to some extent *Activation*. In completing the labs described in this paper, students will apply the skills learned in class to real world design problems achieving the *Problem-centered* principle of instruction. In demonstrating the working labs and discussing what they've learned with the instructors, TAs, and their peers, students will reach the fifth principle of instruction, *Integration*. The goal of these labs is to not only facilitate learning and deepen understanding, but also get sophomore level electrical engineering students excited about the applications of signal processing concepts in everyday life.

The following chapters will detail the labs developed for the new signal processing class. Chapter 2 details a lab centered around chord detection. Students will identify the chord structure of songs from the frequency content of the recording while using filtering and computation techniques to improve the results. Chapter 3 details a musical fingerprinting lab based on the popular app, "Shazam." Students will create unique fingerprints based on frequency content of each song and use them to identify short song clips against a database. Chapter 4 details a lab based on the JPEG image compression protocol. In this lab students will implement a JPEG compression program and compare the effects of different transforms and filters on the results. Chapter 5 details a lab in which students will amplify movements in videos using a phase-based technique. It is based off a paper from Bill Freeman's group at MIT CSAIL [9].

Chapter 2

Chord Detection

For this lab, inspired by the book *Fundamentals of Music Processing* [8], students will analyze a recording of the song to find the underlying chord structure. Students will learn about the frequency content of music and how examining signals in the frequency domain can reveal previously unknown characteristics.

The chord detection process begins with making a chromagram, which describes the pitch profile of the song. For each point in time, the chord closest to the pitches in the chromagram is recorded as the chord being played. Filtering and Markov models can be used to improve the results.

Students will first implement a naive chord detection program, then experiment with filtering and Markov models to improve the program.

2.1 Chromagram

To demonstrate the chord detection process, I've analyzed the 1995 song *Wonderwall* by Oasis.

First, students will make a spectrogram of the song to see the frequency content over time. Students can experiment with different FFT window lengths and different hop sizes. The window and hop size should be chosen such that the feature detection rate is faster than the typical chord change, but not so small that the time samples are subject to bias from irrelevant frequency content. Choosing a feature detection

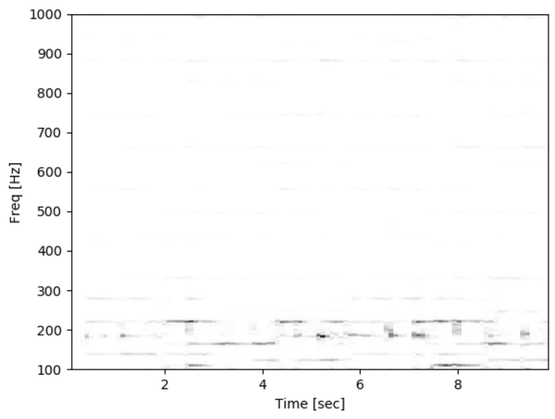


Figure 2-1: Spectrogram of First 10 Seconds of *Wonderwall*

Spectrogram created with FFT window length of 186 milliseconds and an overlap of 93 milliseconds, giving us a feature detection rate of about 10 Hz as suggested by a paper by T. Cho et al [1].

rate much faster than typical chord changes allows results in more accurate time stamps for chord changes. Students can also improve the note detection by zero-padding the spectrogram, which adds more samples to signals in the frequency domain but does not improve the actual resolution. This finer sampling of the frequency spectrum results in sampled values closer in frequency to the precise note frequencies and therefore more accurate note detection.

Audible frequencies range from about 20 Hz through about 20 kHz. However, the useful range for chord detection is only from about 100 Hz to 1000 Hz, as frequencies outside this range rarely occur in music. Only these useful frequencies are displayed in Figure 2-1's spectrogram of the first 10 seconds of *Wonderwall*.

Next, students will write a program which creates a chromagram from the spectrogram. A chromagram is a way of representing the complete pitch class profile of the song. For chord detection the octave of a note doesn't matter, just its pitch. Thus, the chromagram sums together the spectrogram signal sampled at all notes of the same pitch within the useful range. This process is repeated for every pitch value and every point in time for the spectrogram. The resulting chromagram contains the full pitch profile of the song clip as it changes over time. Figure 2-2 shows the resulting chromagram for the 10 second clip of *Wonderwall*.

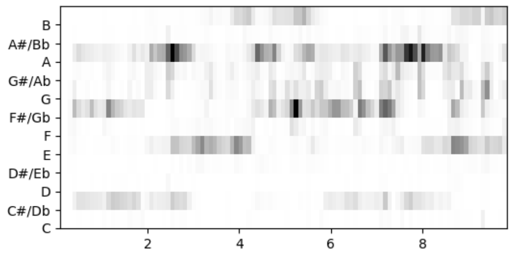


Figure 2-2: Chromagram for *Wonderwall* Clip, Unfiltered

2.2 Determine Closest Chord

Now that the students have a chromagram to show the pitch class profile of the song clip, they will determine which chord is being played at each point in time. They will begin with a naive approach, in which the chromagram at each time step is compared to a bank of vectors representing the different possible chords. The vector determined to be most similar to the chromagram, according to the Pearson correlation coefficient, is labeled as the likely correct chord.

2.2.1 Chord Vectors

Each chord is represented by an array of 12 binary values representing which notes make up the chord. There are 60 possible chords, but many of them are uncommon. To simplify this demonstration, I've chosen to use only 25 chords: the 24 major and minor chords, which are the most commonly used, and the chord Esus, which is a chord in the example song *Wonderwall*. Chords can be added or removed by instructors or students if desired. Figure 2-3 shows the chord vectors for the 25 chosen chords.

2.2.2 Determine Similarity

Students will write a program comparing the chromagram at each timestep to the chord vectors using the Pearson's correlation coefficient. The chord vector with the highest correlation coefficient is recorded as the most similar chord. When the most similar chord is computed for each timestep and combined, the result shows an ap-

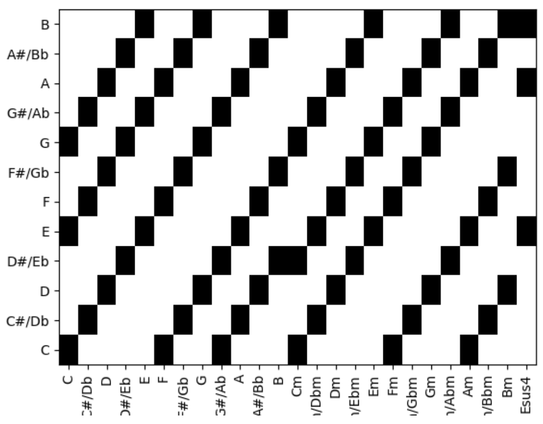


Figure 2-3: Chord Vectors

proximation of the overall chord structure as seen in Figure 2-4. For reference, the actual chord structure of the introduction to *Wonderwall* repeats the chords F#m, A, Esus, and B.

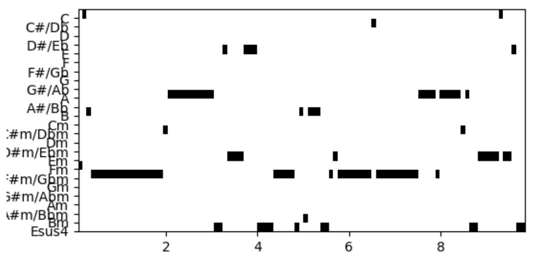


Figure 2-4: Chord Structure from Unfiltered Chromagram

The resulting chord structure for the song clip is still not ideal. It contains a lot of noisy misdetections that do not actually represent the underlying chord structure. However, even this naive approach shows some general trends.The longest cohesive chords in Figure 2-4 are F#m and A, the first two chords in the clip. With this naive approach, the program has some difficulty discerning the Esus chord from the Em chord as they have two of three notes in common. The B chord is also detected, though it is much shorter than the others and difficult to discern from the noise. These common, volatile misdetections are undesirable, but instead of just changing the size of the window to decrease the time resolution, reductions in these misdetections can be accomplished through filtering. In the next part of the lab, students will explore different methods of improvement on this naive approach to chord detection.

2.3 Chromagram Filtering

While Figure 2-4 shows some of the underlying chord structure, the results are still too noisy. Students will improve the results using the following techniques: logarithmic scaling, averaging filters, and the Viterbi algorithm.

2.3.1 Logarithmic Scaling

Human ears discern volume logarithmically, so by adjusting the values of the chromagram to a decibel scale, the values will more closely correspond to the volume the ear itself hears. Students will move their chromagrams to a decibel scale.

2.3.2 Averaging Filter

There are a lot of short, single timestep misdetections in the naive chord detection results in Figure 2-4. One approach to reduce the number of these single step misdetections is to determine the closest chord using a longer time step. To achieve this, students will implement a simple averaging filter in the time domain for each pitch in the chromagram. The students can test how different averaging window lengths affect the resulting chromagram and detected chord structure.

There are some trade-offs when picking the window length of the filter. A shorter window will show better time resolution and as pick up on the more rapid chord changes. However, it will still allow for some noise and incorrect results due to quicker music signatures and arpeggiation, which do not contribute to the underlying chord trend. A longer window length smooths the signal, revealing the longer-term chord trends. However, this is at the cost of detecting more rapid chord changes the accuracy of when a chord changes. A paper by Cho et al [1] recommends a window length equal to about 1.5 seconds when using just the averaging filter to improve the results. This is a pretty wide window as some chords change more rapidly than the window length, and it will drastically reduce the resolution in the time domain of our chromagram. However it does do much better job at finding the underlying chord structure of the song. Figure 2-5 shows this time averaged, logarithmically scaled chromagram.

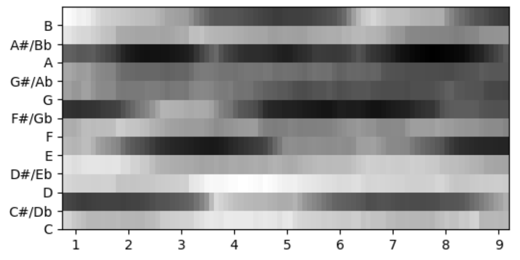


Figure 2-5: Time Averaged Chromagram

For the intro to *Wonderwall*, the resulting chord structure for the time-averaging technique results in correctly detecting three of the four initial chords of the song. It is able to detect the repeated chord structure of F#m, A, then Esus, and has eliminated most of the errors from the previous unfiltered version. With this longer window length, however, it is unable to pick up on the shorter B chord, which should occur around the 5 second mark. Figure 2-6 displays these results.

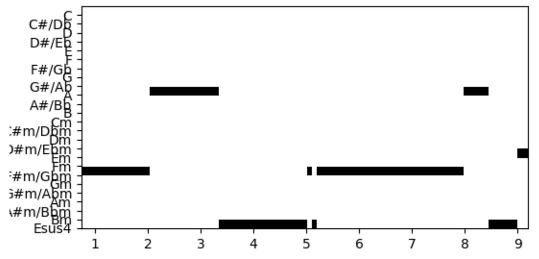


Figure 2-6: Chord Structure from Time Averaged Chromagram

2.3.3 Viterbi Algorithm

Time-averaging has taken care of some of the issues of false chord detection from the original chromagram. However, the long window length has reduced the resolution in the time domain and eliminated some of the shorter chord changes. Ideally, the chord detection program should use a shorter filter window and still get accurate chord changes.

To this end, students will combine the time-averaging method from the previous section with a probabilistic model to get the best of both worlds. After the chords are detected from a smoothed chromagram, the students will use the Viterbi algorithm [2] to determine the most likely sequence of chords from the observed chord structure. The Viterbi algorithm uses Markov models to find the most likely sequence or "Viterbi path" given a string of observed states.

The information needed to run the Viterbi algorithm is (1) the initial belief over the chords, (2) a transition model with the probabilities of transitions from one chord to another between timesteps, and (3) an observation model which shows the probability of detecting a chord given that a chord is being played. Students will have the opportunity to choose and test different probability models on the system.

Probability Models

Students will come up with probability models to use for the Viterbi algorithm and test them on the results. There are several different ways students can design these models, including using existing published chord change data or using some knowledge of music theory. For this example, I've used more basic initial belief and transition models. I've defined the models below.

For the initial belief distribution, I've used a uniform belief over all 25 chords. The following equation shows the initial probability for some chord i:

$$P_i = \frac{1}{25} \tag{2.1}$$

The transition model is the probability of moving from chord i to chord j between timesteps. I've chosen a transition model with an equal probability of moving between chords, but a probability 10 times higher for self-transition. This disincentivizes chord changes. The transition model from chord i to chord j is defined below.

$$P_{ij} = \begin{cases} \frac{1}{34} & \text{for } i \neq j \\ \frac{10}{34} & \text{for } i = j \end{cases} \tag{2.2}$$

The observation model is the probability of detecting some chord i given that some chord j is being played. For this model, the Pearson correlation coefficient between vectors was chosen as the chord detection value. Correlations of 0 or below (corresponding to zero common notes) were set to a low nonzero value, in this case 0.02. The detection probability distribution for each played chord is then normalized. The resulting observation model has diminishing probabilities for false detections assigned to chord pairs with two, one, and zero common notes, respectively.

2.4 Resulting Chord Progression

The chord transition model and the averaging filter window length, L, can be manipulated by the students to determine the combination with the best performance. According to the Cho et al paper [1], the ideal value for the averaging-filter when combined with the Viterbi algorithm is $L = 3$. The results of the complete chord detection algorithm on this 10 second sample of *Wonderwall* are shown in Figure 2-7. The results from this hybrid method show improvements over the averaging-filter only

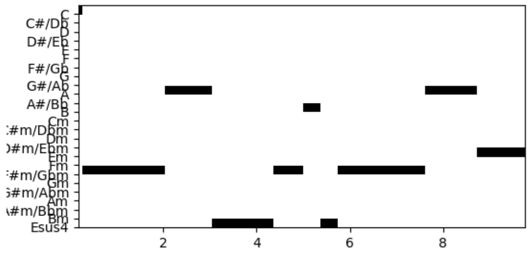


Figure 2-7: *Wonderwall* Chord Structure Post Viterbi Algorithm

Averaging filter window width N=3

method. The time stamps of the chord changes are more accurate and it picked up on the short B chord. It's still not perfect; the B chord detection is sandwiched between short, incorrect F#m and Esus detections, but it is the most accurate chord structure result yet and does not include the short, single step misdetections characteristic of the naive chord detection approach in Figure 2-4.

Chapter 3

Musical Fingerprinting

The popular app "Shazam" identifies songs from a short, recorded clip, by creating a fingerprint of the clip and matching it to a database of songs [10]. In this lab students will match short song clips - both perfect and distorted - to the correct time in the full song and then to a song within a database. Through this lab students will learn about computational tools and combine them with the signal processing tools they've already learned to great effect.

The general procedure as outlined by the paper by Avery Wang [10] is as follows: A unique fingerprint for the song clip is created based on the locations of the frequency peaks in time. A hash value is generated for each point in the fingerprint, and compared to a hashed database of songs. The reverse lookup returns the location of the clip in the database.

3.1 Create Fingerprint

To create the database, a fingerprint is generated for each song and song clip to make it uniquely identifiable. To demonstrate this process, students will create a fingerprint for a short song segment. The following example uses the song *Stars Dance* by Selena Gomez.

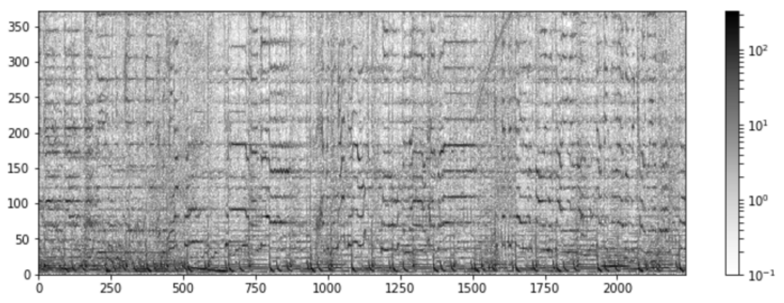


Figure 3-1: Spectrogram of *Stars Dance* Clip

Spectrogram created using FFT window-length of 2048 samples (about 92 ms) and hop size of 256 samples (about 11 ms)

3.1.1 Spectrogram

The first step of the fingerprinting process is to create a spectrogram on the song. Students can experiment with the effects of different window-lengths and hop sizes. However, it is important that the spectrogram values remain consistent throughout the implementation of the lab to ensure that the resulting fingerprints will match up. Figure 3-1 is the spectrogram of the song clip.

3.1.2 Constellation

A song can be distinguished by its unique frequency profile. A frequency profile is like a constellation of a song; the positions of the frequency peaks in time create a signature of the song. Notably, the position of these peaks holds even in recorded clips distorted by some background noise, although the magnitude of the peaks relative to baseline activity may be affected.

Students will write a program to create this "constellation" of peaks. In order to avoid spurious peaks throughout the spectrogram, students can find the local maximum peak above some threshold within a small rectangular region of the spectrogram. Figure 3-2 is an example of this process performed on a single rectangle. The size of this rectangle is significant: too large, and the constellation will be too sparse, leaving out potentially useful identifying structures; too small, and the constellation will potentially pick up false peaks as well as make the future matching steps more

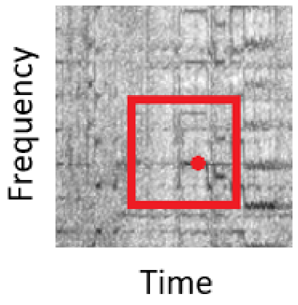


Figure 3-2: Finding the Local Maximum in a Rectangle

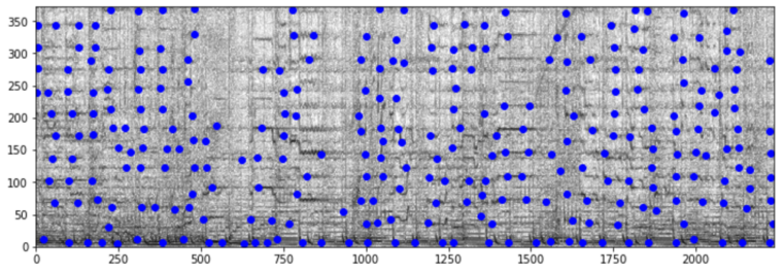


Figure 3-3: Constellation and Spectrogram of Song Clip

Constellation generated using a rectangle width of 60 samples (about 700 ms), and a height of 40 frequency bins (about 430 Hz).

computationally complex. In this step, students can select a rectangle size they deem appropriate. For each rectangle of the spectrogram, the largest peak above a certain threshold is recorded, and the resulting constellation has been plotted over the spectrogram to show how they match up. Figure 3-3 shows the resulting constellation for the song clip.

3.2 Matching

In this next section of the lab students will use the generated constellations to match the song clip to the correct time in the full song.

A full song constellation must be generated in order to match the clip to the correct time in the full song. Figure 3-4 is the constellation of the full song, $C(D)$. The constellation of a short time slice, or "query", is compared to the constellation

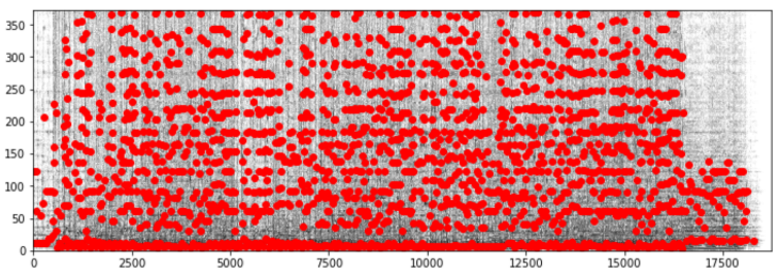


Figure 3-4: Full Constellation of *Stars Dance*

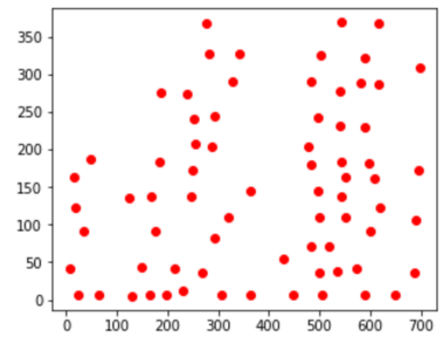


Figure 3-5: 1.5 Second Song Query

Query length is 1.5 seconds.

as a whole. Figure 3-5 shows the query constellation.

To identify where a query clip occurs within the full song, the query constellation is translated along the length of the full constellation. For each time offset, the number of constellation points that match is recorded as a score. The offset with the highest score is the most likely time location of the query.

Students will use this technique to match the query to the correct location in the full song. Figure 3-5 shows the example query matched to the the full song as seen in Figure 3-4. Figure 3-6 shows the results of this procedure. The top graph shows the query constellation in blue matched against the full constellation in red. The bottom graph shows the score on the y axis for each offset on the x axis, with the highest speak score corresponding to the optimal offset. For the query in Figure 3-6 the correct time location was successfully found. The maximum score for the location was 30 matching points, far higher than any other time step.

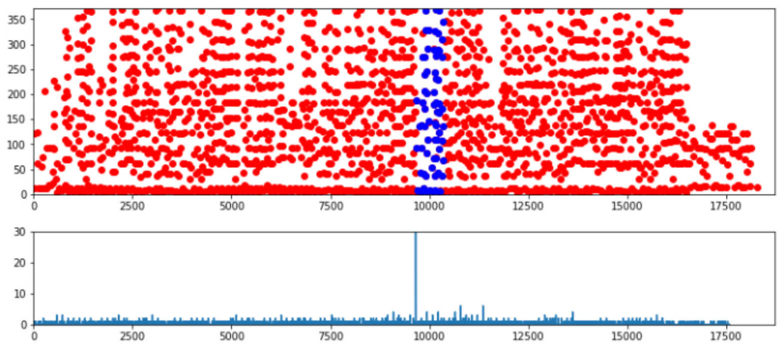


Figure 3-6: Time Matched Query for Original Clip

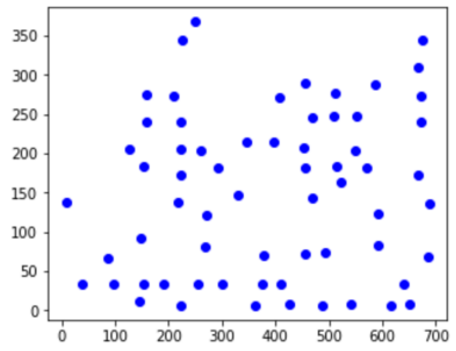


Figure 3-7: Distorted Song Query

3.3 Matching a Distorted Song Clip

The fingerprinting and matching procedure correctly placed the song clip within the song, but this clip was an exact copy of the original song. What happens if the clip to be identified isn't perfect, such as they type one's phone might record in a car or coffee shop? Students will perform the same fingerprinting and matching procedure on a distorted song clip to see if it still works.

Using the same spectrogram settings and rectangle size as the previous section, a constellation for the distorted query was created and is shown in Figure 3-7. This query was matched against the undistorted full song constellation as seen in Figure 3-4 using the same procedure as described in the previous section. The procedure successfully matched the distorted query to the appropriate location in the full song and the results can be seen in Figure 3-8. Once again, the top graph shows the query constellation in blue matched against the full constellation in red. The bottom graph

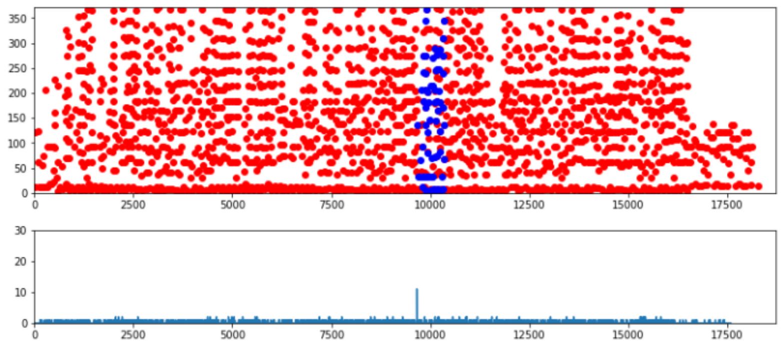


Figure 3-8: Time Matched Query for Distorted Clip

shows the score on the y axis for each offset on the x axis, with the highest speak score corresponding to the optimal offset.

The maximum score for the distorted query is only 10, which is much lower than the undistorted query's 30 matching points. However, these 10 matched points are still significantly higher than the scores for all other offsets. Despite the distorted, noisy nature of this song clip, the underlying frequency structures were identifiable.

3.4 Query Optimization

The sliding constellation method works well when comparing the query to the single song to determine the offset. However, when comparing an unknown song clip query to a larger database of songs, the process of comparing every constellation point in the query (size M) to every constellation point in the database (size N) is computationally expensive, and increases as $M \cdot N$. The "Shazam" app manages to search a large database of songs in a matter of seconds.

To reduce the computational complexity of the search students will implement a simple hash function. A hash value h is computed for every constellation point (n, k). Given a well chosen hash function, the constellation points are approximately evenly spread over all hash values. A lookup table of the hash values reduces the point values to be matched from N values to about $\frac{N}{L}$ values. This reduces the computational complexity from $M \cdot N$ to $\frac{M \cdot N}{L}$, where N is the number of constellation points in the database, M is the number of constellation points in the query, and L is the number of hash values.

Students will likely initially implement a simple hash value of $h = k$. This results in L number of hash values, h, corresponding to the number of frequency bins in the constellation. To improve the speed of the hash function the number of hash values L should be increased by using more complex hash definitions. Students will be aware of this from the $\frac{M \cdot N}{L}$ relationship, and should they desire they can design a better hash function.

Chapter 4

JPEG Compression Lab

Students interact with compressed images on the internet almost every day, but may not have thought about the signal processing techniques used to compress these images. The purpose of this lab is to teach two-dimensional image processing techniques and reinforce the relationship between the spatial and frequency domains through the context of JPEG image compression. In this lab, students will implement a program performing JPEG compression on an image, compare the performance of different transforms in the context of JPEG image compression, and determine the best kind of filter to use for the lossy stage of the process.

The general procedure for JPEG compression is as follows: the image is first divided into small blocks of pixels and transformed to the frequency domain. The less important high-frequency data is attenuated by a low pass filter. The block of reduced frequency information is encoded and returned. The procedure is reversed to decompress the file and recreate the image.

4.1 Perform Compression/Decompression

The first task students will complete is to implement a JPEG compression program and use it to compress an image of their choice. I've demonstrated the process using the image in Figure 4-1, taken by David Hu. Usually, in standard JPEG compression, the image is converted to YIQ format and the luma and chrominance layers are

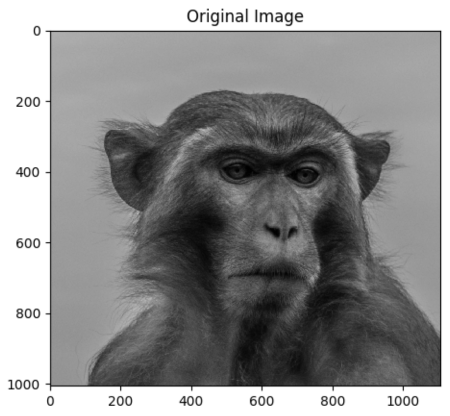


Figure 4-1: Original Uncompressed Image

compressed separately using different filters. However, for the sake of simplicity this example uses a flattened image which reduces it to a single greyscale layer.

4.1.1 Compression

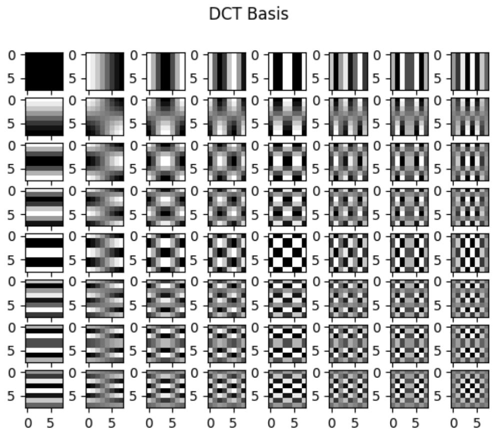


Figure 4-2: Discrete Cosine Transform Basis

The DCT results in a block in the frequency domain with the lower frequency components in the upper left corner and the higher frequency components in the lower left corner.

To compress the image to JPEG format, students first will write a program that separates the image into boxes of 8 by 8 pixels. They will then convert each box to from the spatial domain into the frequency domain using the two-dimensional Discrete

Cosine Transform, which is defined by the following formula.

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} cos\left[\frac{\pi}{N_1}\left(n_1 + \frac{1}{2}\right)k_1\right] cos\left[\frac{\pi}{N_2}\left(n_1 + \frac{1}{2}\right)k_2\right] \tag{4.1}$$

To visualize the DCT of an 8 by 8 pixel block I've generated a DCT basis as seen in Figure 4-2. Any 8 by 8 combination of pixels can be reconstructed by a combination of these functions. JPEG compression uses the Discrete Cosine Transform instead of the more common Fast Fourier Transform, which students will be more familiar with at this point in the class. Students will explore the reason for the use of the DCT over the FFT as described in Section 4.2.

Now that the block of pixels is entirely in the frequency domain, students will implement the lossy part of the compression. The human eye itself acts as a low pass filter [7], so eliminating the higher frequency data reduces the information necessary to represent the image while minimizing the perception of change to the human eye. The following quantization table is the default luminescence JPEG standard table corresponding to 75% compression [4].

$$T_b = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 49 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix} \tag{4.2}$$

Each element of the 8 by 8 pixel frequency domain block is divided by the corresponding element in the quantization matrix, then rounded to the nearest integer. The larger divisors in the lower right corner of the block attenuate the higher frequencies and the lower divisors in the upper right corner preserve more of the lower frequency information. To increase or decrease the compression of the image, this standard table can be adjusted via the quality factor Q. The scale factor S is determined based on the quality factor Q as follows

$$S = \begin{cases} \frac{5000}{Q} & \text{if } Q < 50 \\ 200 - 2 \cdot Q & \text{if } Q \geq 50 \end{cases} \tag{4.3}$$

This scale factor S is then applied to each element of the base matrix from equation 4.2.

$$T_S[i] = \left\lfloor \frac{S \cdot T_b[i] + 50}{100} \right\rfloor \tag{4.4}$$

The elements are rounded down to the nearest integer value. Students can compare the effects of compression with different quality factors on their images. A comparison of quality factors on image quality can be seen in Figure 4-6 towards the end of this section. If unspecified, all further examples in this chapter will use a quality factor of 50 corresponding to the unchanged quantization table from Equation 4.2.

This quantization step is where the information is lost. For the higher frequencies the information is completely lost and the coefficients become 0. To demonstrate the results of this step, a block of pixels (from the example image) before and after quantization is shown in Figure 4-3.

After quantization, all of the nonzero coefficients are in the upper left corner. The block of pixels can be deconstructed such that the nonzero low frequency coefficients are at the beginning of a list, and a long trail of zeros from the high frequency coefficients is at the end. When using an encoding method such as Huffman or run-length encoding on this deconstructed block, these repeated characters result in a much smaller file size. The order of pixels in this deconstruction is the following.

$$\begin{pmatrix} 0 & 1 & 5 & 6 & 14 & 15 & 27 & 28 \\ 2 & 4 & 7 & 13 & 16 & 26 & 29 & 42 \\ 3 & 8 & 12 & 17 & 25 & 30 & 41 & 43 \\ 9 & 11 & 18 & 24 & 31 & 40 & 44 & 53 \\ 10 & 19 & 23 & 32 & 39 & 45 & 52 & 54 \\ 20 & 22 & 33 & 38 & 46 & 51 & 55 & 60 \\ 21 & 34 & 37 & 47 & 50 & 56 & 59 & 61 \\ 35 & 36 & 48 & 49 & 57 & 58 & 62 & 63 \end{pmatrix} \tag{4.5}$$

For this implementation of JPEG compression I've used run-length encoding instead

$$\begin{pmatrix} -550.63 & -29.20 & 19.90 & -10.73 & 1.13 & -0.35 & 4.61 & 1.41 \\ -21.59 & -10.28 & 3.85 & 8.64 & 3.23 & 4.45 & 2.98 & -2.18 \\ -4.97 & 10.10 & 19.02 & 7.71 & -0.11 & 0.89 & -1.14 & -2.21 \\ 3.20 & 20.98 & -10.41 & 6.06 & -1.96 & -6.45 & -4.30 & -0.93 \\ 5.37 & -7.60 & 4.91 & 7.45 & 4.63 & -3.76 & 1.08 & -0.99 \\ 1.35 & 3.29 & 6.25 & -14.10 & 2.78 & -3.67 & -2.67 & 2.33 \\ 1.38 & -4.54 & 0.61 & -1.63 & 0.72 & 5.36 & -2.02 & -0.65 \\ 0.45 & 0.17 & -1.65 & -3.30 & -1.30 & -1.73 & -1.01 & -1.11 \end{pmatrix}$$

(a) Before Quantization

$$\begin{pmatrix} -34 & -3 & 2 & -1 & 0 & 0 & 0 & 0 \\ -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(b) After Quantization

Figure 4-3: Quantization in the Frequency Domain, $Q = 50$

Float values in the "before quantization" array are displayed to only 2 decimals here.

of Huffman encoding but students can choose which encoding procedure they prefer. The quantized block from Figure 4-3 was deconstructed and encoded using run-length encoding and the resulting compressed list is below.

```
[(-34, 1), (-3, 1), (-2, 1), (0, 1), (-1, 1), (2, 1), (-1, 1), \\
(0, 1), (1, 1), (0, 2), (1, 2), (0, 51)]
```

Students will implement a program to perform this process on every 8 by 8 block of pixels and add to a long list of characters which represents the entire compressed JPEG image. As I implemented the current version of this lab, this list is not serialized into a file, though at this point in the process it could be. Instead, the compression function returns the compressed list as well as the dimensions of the original image.

4.1.2 Decompression

Now that the image has been compressed, the file size is much smaller, which makes it easier to store and transfer. However, one can't just look at this compressed list and

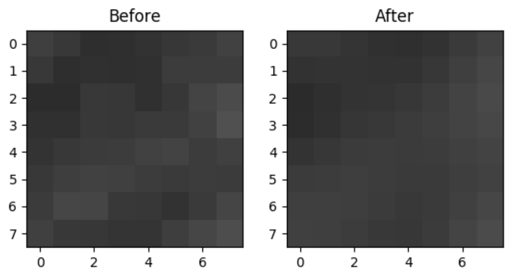


Figure 4-4: Block of pixels before and after compression

see an image. In this part of the lab, students will write a decompression program to convert the compressed list back into an array of pixels.

First they will reverse the encoding process used in the compression step (in this example run-length encoding), resulting in a long list of integers corresponding to frequency coefficients. Every 64 coefficients is one 8 by 8 block of pixels. The 8 by 8 block is reconstructed from these 64 coefficients using the same ordering as shown in equation 4.5. The resulting block is then multiplied elementwise by the same quantization matrix as was used in the compression stage. The block is then converted back to the spatial domain using an inverse two-dimensional Discrete Cosine Transform. Due to the compression process, the resulting block will be different from the original block, but given a reasonable choice of quality factor Q, it will be at least recognizable, if not nearly identical the original. See Figure 4-4 for a side by side comparison of a block of pixels before and after compression.

This process is performed for every 8 by 8 block of pixels, and the blocks are arranged such that the final image matches the original image's dimensions. A side by side comparison of the full image before and after compression is shown in Figure 4-5. A comparison of images compressed using two vastly different quality factors, $Q = 50$ and $Q = 10$ respectively, is shown in Figure 4-6. The image has been zoomed in to better see the difference.

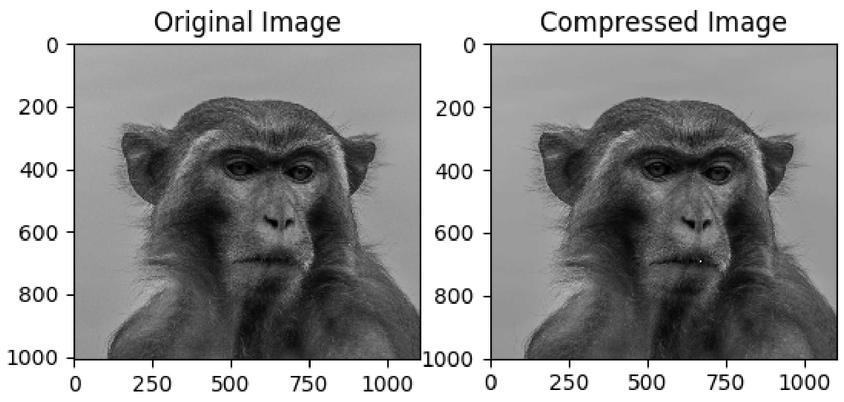


Figure 4-5: Complete Image Before and After Compression

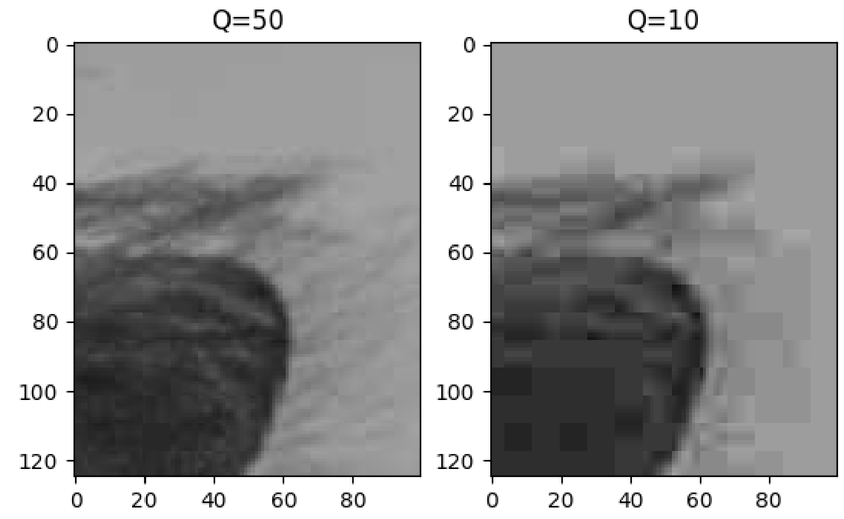
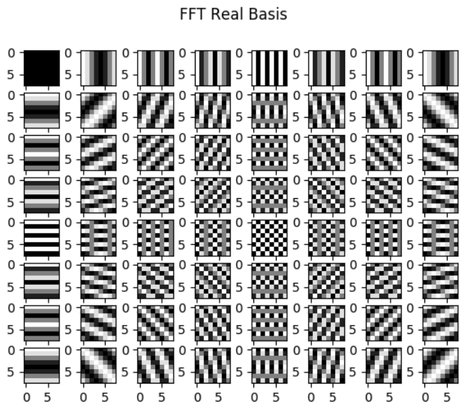


Figure 4-6: Comparison of Images Compressed using Different Quality Factors


(a) Real

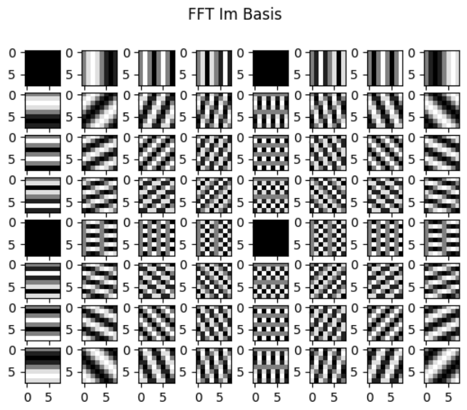

(b) Imaginary

Figure 4-7: FFT Basis

4.2 Compare DCT and FFT

The Discrete Cosine Transform is the transform used for JPEG compression, but at this point in the course students will be much more familiar with the Discrete Time Fourier Transform or the Fast Fourier Transform. However, the DCT is superior to the FFT for this particular application. In this next part of the lab students will compare JPEG compression using the DCT and the FFT to determine these reasons for themselves. One way to compare these transforms is to look at the basis for each. The basis for the DCT can be see earlier in Figure 4-2. The basis for the FFT can be seen in Figure 4-7.

Because the image is an entirely real signal, it can be entirely represented by even functions (e.g. cosines). The DCT decomposes the image into an entirely cosine basis, so the 8 by 8 block is represented by only 64 real coefficients. The FFT is constructed of both even and odd functions, so the same 8 by 8 pixel block must be represented by 128 coefficients: 64 real and 64 imaginary. The goal of compression is to represent the image with the least space possible, so twice as many coefficients to represent the same quality image is undesirable.

To demonstrate the superiority of the DCT for this application, students will compare the outcomes of JPEG compression using both the DCT and the FFT with the same number of coefficients. To make sure the lossy step is fair to both the DCT

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 4-8: Quantization Matrix for DCT Version

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(a) Real

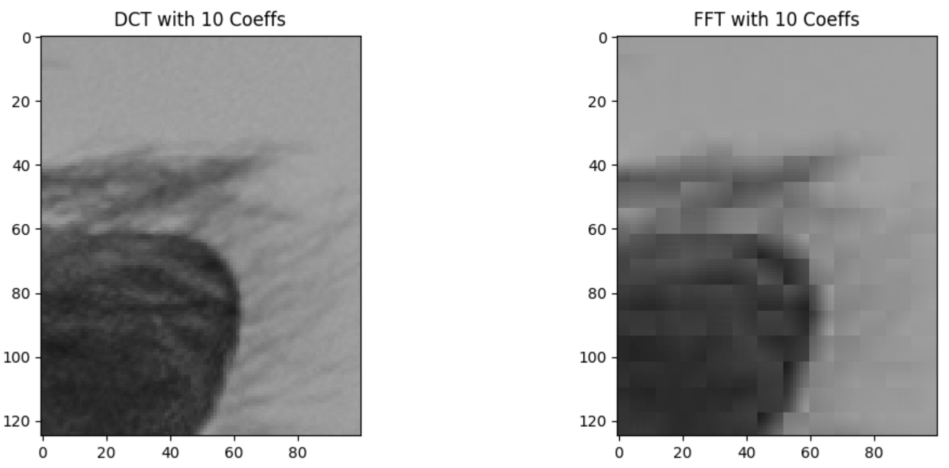
$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(b) Imaginary

Figure 4-9: Quantization Matrices for FFT Method

and FFT techniques, instead of dividing by a quantization matrix and rounding off to the nearest integer, each block will be multiplied by a matrix entirely composed of zeros and ones, eliminating the higher frequency content and keeping the lowest frequency content. For this example I used the 10 lowest frequency coefficients. The matrix used for the DCT method can be seen in Figure 4-8. The matrices used for the FFT method can be seen in Figure 4-9.

The rest of the compression and decompression process remains the same for both the DCT and FFT methods. The resulting photo comparison can be seen in Figure 4-10. The image has been zoomed in to better see the difference. The DCT image is closer to the original image than the FFT image using the same amount of coefficients.



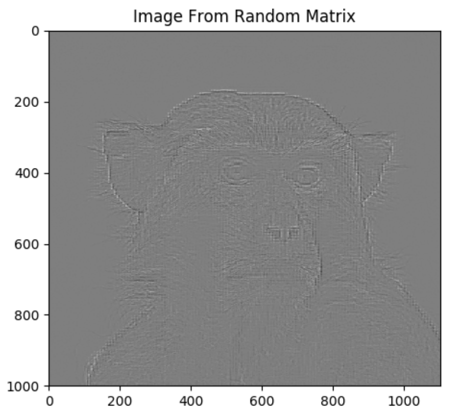
(a) DCT (b) FFT

Figure 4-10: Final Compressed Images With 10 Coefficients

4.3 Optimize Filter

The quantization matrix used in JPEG compression operates as a low pass filter, attenuating higher frequencies and passing the lower frequencies. The human eye itself operates as a low pass filter [7], so this results in an image that appears closest to the original. In this section of the lab, students will discover for themselves that the low pass filter results in the best image for the number of coefficients saved. To reach this conclusion, students will compare JPEGs compressed using different filtering methods and determine which results in the best image.

Students will be presented with side by side images compressed using the DCT compression method and randomized quantization matrices. These 8 by 8 pixel quantization matrices are generated with 16 randomly placed '1s' and the rest '0s'. This ensures that the same number of coefficients is used to represent either image. These quantization matrices are multiplied by each pixel block in the lossy step of the compression. Of the two resulting images, students will choose which one looks closer to the original image. This image is saved and a new image compressed using a newly generated random quantization matrix is compared to it. After a few dozen image comparisons, the resulting "best" image will have been generated from a matrix with



$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(a) Generated Image (b) Matrix Used to Generate Image

Figure 4-11: Image Generated from Random Quantization Matrix

most of the ones in the lower frequency coefficients. This demonstrates that the low pass filter is the ideal filter for JPEG image compression. An example of an image generated from a random matrix and the corresponding matrix can be seen in Figure 4-11.

Chapter 5

Phase Based Movement Magnification in Videos

This lab, based on a paper from Professor Bill Freeman's group at MIT CSAIL [9], explores Fourier techniques in three dimensions. Through this lab, students will learn about the relationship between phase and the spacial domain. The method generally works by amplifying the change between each frame, which magnifies the motion of the video. To demonstrate this, the students will amplify motion in simplified frames as well as actual videos.

5.1 Simplified Versions

5.1.1 Displacement and Phase Relationship

Before delving into the implementation of phase magnification in one dimension, students must have a firm grasp of the relationship between spacial translation and phase. A one dimensional discrete signal has the following transform.

$$x[n] \Leftrightarrow X(\Omega) \tag{5.1}$$

A one dimensional discrete signal with some spacial displacement n_0 has the following transform.

$$x[n - n_0] \Leftrightarrow X(\Omega)e^{-j\Omega n_0} \tag{5.2}$$

The translation in the spacial domain corresponds to a change in phase in the frequency domain. Conversely, a change in phase in the frequency domain corresponds to a shift in the spacial domain. Students will take advantage of this relationship by magnifying the phase of the signal causing a corresponding magnified shift in the spacial domain.

5.1.2 Movement Magnification in One Dimension

To demonstrate this process, students will first perform movement magfication on a one-dimensional "video." Students will first choose the one-dimensional image they wish to translate. For this demonstration I've used a simple sinusoid with one cycle as the image function where N is the number of samples per unit (ie position $x = \frac{n}{N}$).

$$x[n] = cos\left(\frac{2\pi n}{N}\right) \Leftrightarrow X(\Omega) \tag{5.3}$$

A small time-dependent displacement is added in the spacial domain changing the function as follows

$$x[n + \delta(t)] = cos\left(2\pi\frac{n + \delta(t)}{N}\right) \Leftrightarrow X(\Omega)e^{j\Omega\delta(t)} \tag{5.4}$$

I've defined $\delta(t)$ as the following.

$$\delta(t) = 0.005sin(8\pi t) \tag{5.5}$$

Students will plot this time-dependent shifting frame and use a slider to adjust the time t to see the small spacial translation. Figure 5-1a shows the image at time $t = 0$. In moving the time slider, students will see an extremely small, barely perceptible sinusoidal movement of the image.

Next, students will perform phase magnification on their time-dependent images. First they will need to isolate the initial phase shift. The transformed signal with time-dependent displacement is

$$X(\Omega) = \sum_{n=-\infty}^{\infty} A_n e^{-j\Omega(n+\delta(t))} \tag{5.6}$$

The phase of this is

$$\phi = -\Omega(n + \delta(t)). \tag{5.7}$$

The full phase shouldn't be magnified, only the phase corresponding to the time-dependent shift $\delta(t)$. To do this, the DC component of the phase, $\Omega \cdot n$, must be filtered out. Students should remove this DC component by subtracting the phase of the signal at rest as follows.

$$\Delta_\phi = -\Omega(n + \delta(t)) - (-\Omega \cdot n) = -\Omega \cdot \delta(t) \tag{5.8}$$

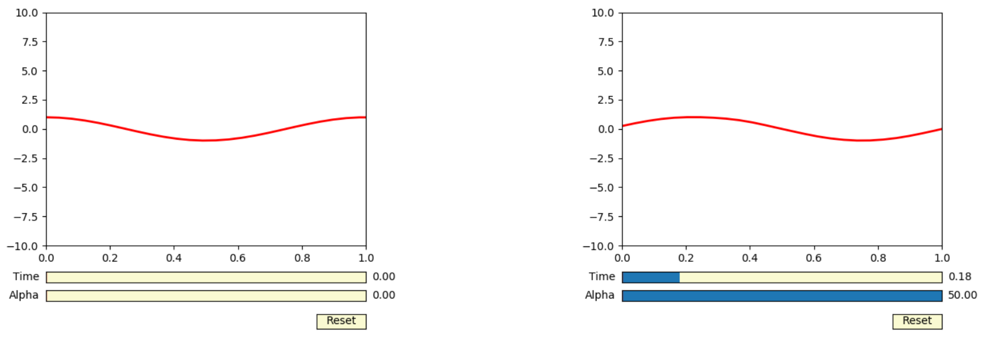
The resulting change in phase, $\Delta_\phi = -\Omega \cdot \delta(t)$, is directly proportional to the time-dependent shift function. To amplify the magnitude of this shift, students will multiply the phase difference Δ_ϕ by some magnification factor α, and add the new phase shift back into the original function.

$$X_\alpha(\Omega) = \sum_{n=-\infty}^{\infty} A_n e^{-j\Omega(n+(1+\alpha)\delta(t))} \tag{5.9}$$

Reconstructing the signal results in the following.

$$X_\alpha(\Omega) \Leftrightarrow x[n + (1 + \alpha)\delta(t)] \tag{5.10}$$

The phase amplification has resulted in a time-dependent shift of magnitude *alpha* greater than the original function. Students will implement this phase-magnification process on the one-dimensional images they used in Figure 5-1a. In addition to a slider corresponding to time t, students will have a slider corresponding to the desired magnification factor α. They can use these sliders to see how different values



(a) Rest State (b) Maximum Displacement with $\alpha = 50$

Figure 5-1: One Dimensional Motion Magnification

of α affect the movement of the image in time. Figure 5-1b shows the maximum displacement of the image for the $\alpha = 50$ case. The displacement is much greater than that of the original image (corresponding to the $\alpha = 0$ case).

Now that the students have examined how this phase based motion magnification affects movement in one dimension they will explore simple two dimensional frames before moving on to actual footage.

5.1.3 Simple Two Dimensional Frames

In two dimensional frames the displacement function has both a time and spacial component, $\delta(x, t)$. To explore simple movements in simple frames, students will first perform the phase magnification process is performed on 16 by 16 pixel frames with sinusoidal translations.

The process for performing simple phase based motion magnification process on these smaller frames is as follows. First, a reference frame is chosen, and the phase of this reference frame is calculated. For each frame in the video, the phase is calculated, and the phase of the reference frame is subtracted from it. This step acts as a bandpass filter and removes the DC component of the phase signal as seen in the one dimensional example (Eq 5.8). This phase difference is multiplied by the chosen α, and the new frame is constructed with the new phase just as is done in the one dimensional example (Eq 5.9).

One Directional Movement

First students will work on a video with the property of one-directional translational movement. Two separeted pixels are translated up and down repeatedly. The first three frames for the unmagnified video are in the top row of Figure 5-2. The middle row of the figure shows the phase of each frame, and the bottom row shows the phase difference of each frame.

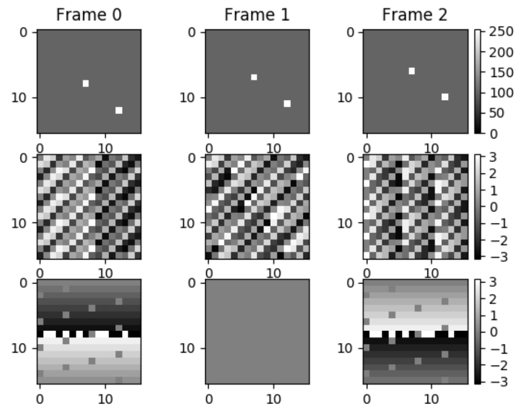


Figure 5-2: Unmagnified Frames for One Directional Translation

Top row shows the frame, middle row shows the phase, bottom row shows the phase difference from the reference frame (frame 1).

With a value of $\alpha = 3$ (small because the dimensions of each frame are only 16 by 16), the resulting magnified frames are in Figure 5-3. For the one-directional motion case, the magnification process results in a perfect movement magnification with no interference.

Two Directional Movement

Instead of using two pixels moving in unison, students will next address two movements in different directions. Figure 5-4 shows two pixels moving between frames with one moving vertically and the other moving horizontally.

The phase based motion magnification process is performed on this video with an amplification value of $\alpha = 3$ once again, and the resulting frames are in Figure 5-5. The resulting frames do not show the desired magnification. Instead, interference

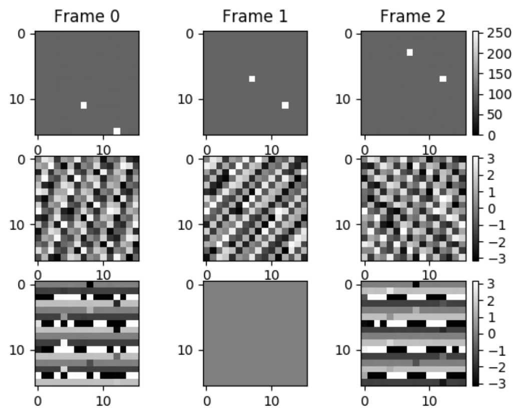


Figure 5-3: Magnified Frames for One Directional Translation with $\alpha = 3$

Top row shows the frame, middle row shows the phase, bottom row shows the phase difference from the reference frame (frame 1).

from the attempted phase-magnificaton of opposing movements affects every pixel in the frame. In practice, videos tend to contain movements of many different directions and frequencies, so to successfully magnify the motion of a video some change must be made to the process. As the students move on from these simpler frames to real video clips, they will have to implement a means of isolating the different motions in order to get meaningful results.

5.2 Steerable Pyramid Version

The first motion isolation technique I explored in creating this lab was the method recommended by Bill Freeman's group's paper [9], the complex steerable pyramid decomposition. Students will not implement the method described in this section and will instead implement the method described in Section 5.3.

Complex steerable pyramid filters "decompose an image according to spacial scale, orientation, and position [9]." The scale and position components of the filters isolate the motions, approximating the one-directional movement case. After performing the phase magnification process on these filtered frames, they are combined to construct a new phase magnified frame with little noise or interference from opposing motions. One level of a complex steerable pyramid filter bank is shown in the frequency domain

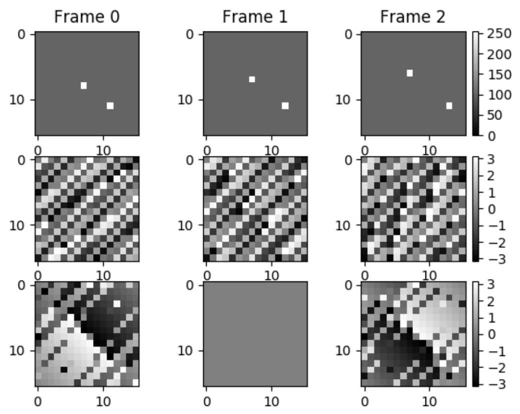


Figure 5-4: Unmagnified Frames for Bidirectional Translation

Top row shows the frame, middle row shows the phase, bottom row shows the phase difference from the reference frame (frame 1).

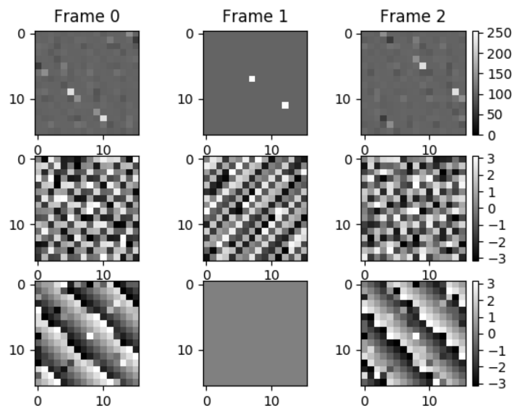


Figure 5-5: Magnified Frames for Bidirectional Translation with $\alpha = 3$

Top row shows the frame, middle row shows the phase, bottom row shows the phase difference from the reference frame (frame 1).

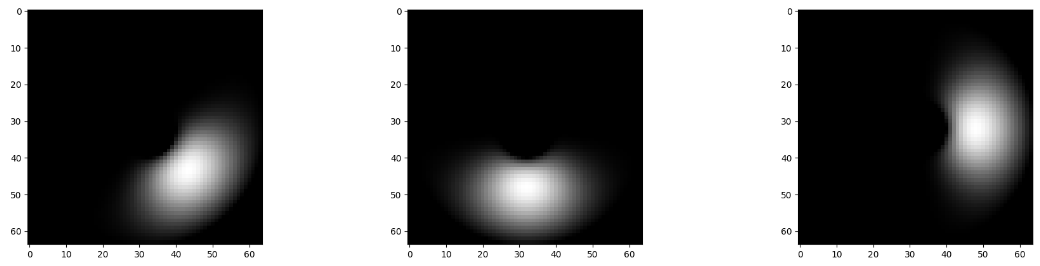


Figure 5-6: Steerable Pyramid Filters for One Level

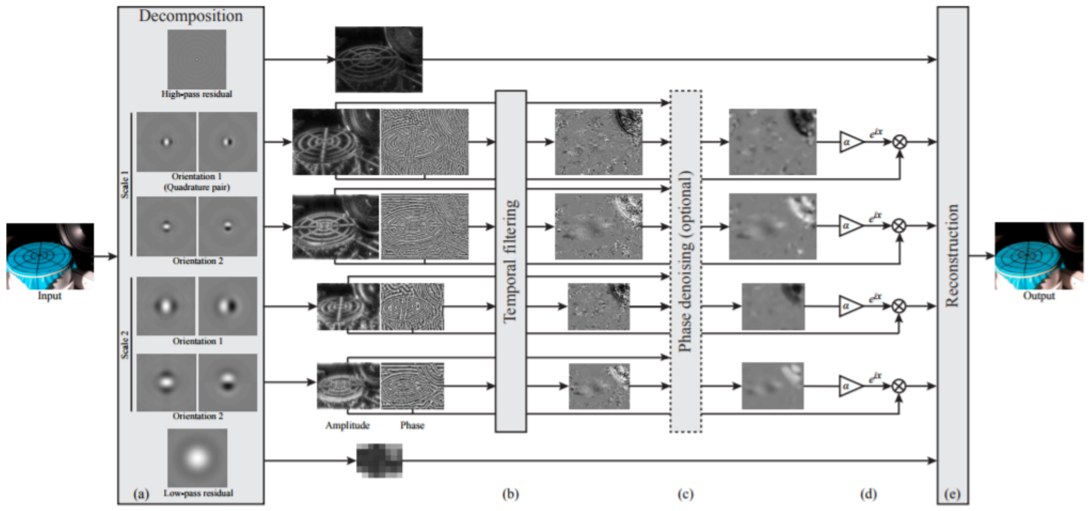


Figure 5-7: Diagram of Phase Based Motion Magnification from CSAIL [9]

in Figure 5-6.

This program was mainly constructed by directly converting the MATLAB code release from the CSAIL Video Magnification Group [9] to Python.

Figure 5-7 is a diagram of the methodology used by Bill Freeman's group. Each frame is decomposed using a complex steerable pyramid filter bank as seen in Figure 5-7 section (a). The phase is then extracted from each frame component, and temporally filtered as seen in Figure 5-7 section (b). This removes the DC component of the phase resulting in a phase difference, and also selects which frequencies of movement should be magnified. The phase difference of each frame component is added back to the original phase to construct the new magnified frame component as seen in Figure 5-7 section (d). Finally, the steerable pyramid is collapsed to construct the new full frame. This process is performed for every frame in the video.

A video of a baby breathing from the CSAIL lab's code release was magnified using this program [9]. The program took about 45 minutes to run on a laptop, and the resulting video showed a noticeable amplification in movement.

5.3 Gaussian Filter Version

The complex steerable pyramid method is the movement localization method recommended by the paper [9], but complex steerable pyramid filters may be difficult to understand for sophomore-level students. Similar results can be reached using a simpler technique students will have already encountered: the Gaussian filter.

Generally speaking, within the small localized area of a video, movements will likely be due to the motion of the same object, and therefore likely be in the same direction. Students will take advantage of this by localizing the region in which they implement the phase magnification algorithm.

The simplest way to accomplish this would be to run the algorithm on square subsections of the larger frame. This would localize the motion magnification to a much smaller area and likely eliminate much of the interference from conflicting movements. Students may implement this naive localization technique first. However, using discrete sections will not allow movements to cross the border of each section, leading to strange edge effects. Using a Gaussian window smoothly attenuates the signal outside of a small region. This localizes the signal to a small region just as breaking the frame into squares would, but with the smooth attenuation of the signal it will not result in strange edge effects. Figure 5-8 shows a frame with a Gaussian window applied to it. This will isolate the phase magnification procedure to only movements in this region.

Students will use Gaussian filters centered at different points around the image (one standard deviation apart) to decompose each frame. Each Gaussian component is then separately phase magnified and recombined to create the complete magnified frame. A summary of this process is below.

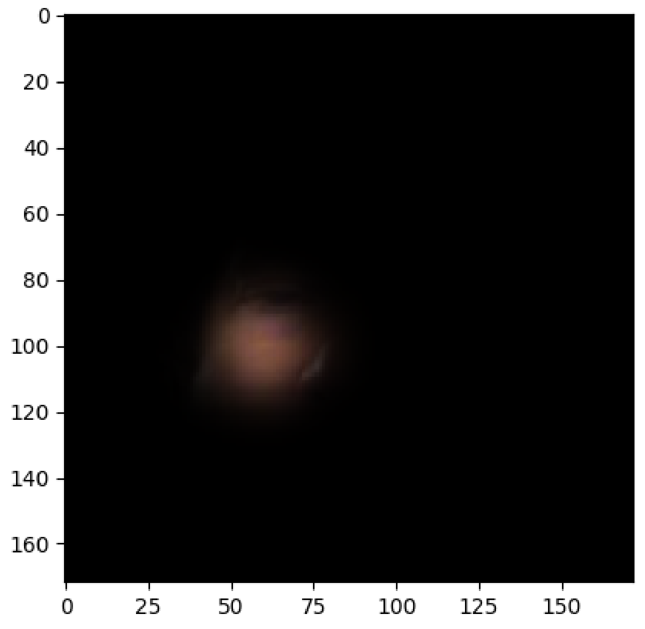


Figure 5-8: Gaussian Windowed Frame

1. Gaussian windows applied to frame to create localized components.

2. Phase of each windowed frame is calculated, and the reference frame's phase is subtracted from them, which eliminates the DC signal and results in the phase difference. The phase difference is unwrapped and smoothed using a moving average filter.

3. The magnified phase difference is added back to the original windowed-frame phase and the new Gaussian frame component is generated.

4. Finally, the complete frame is generated by adding each Gaussian component.

Students will perform this process for every frame in the video.

I tested this Gaussian localization phase magnification method on a video with a shape of 172 by 172 pixels and 81 Gaussian windows. The program took about 10 minutes to run. The resulting video showed a noticeable magnification of movement.

Chapter 6

Conclusion

The labs described in this paper will be incorporated into a new sophomore-level signal processing class at MIT offered tentatively in the Spring 2018 term. The educational benefits of these labs will not be measured until the class is offered but the hope is that the applications-based approach will improve upon current signals classes. With the addition of these applications-based labs, the real life relevance of signal processing will be made clear while allowing students to apply their new found skills to interesting problems. This *Problem-Centered* approach is one of the key tenants of M. David Merrill's five First Principles of Instruction [5] and will hopefully increase the impact of signals education at MIT.

These labs have been designed to improve student understanding of signal processing techniques and allow them to apply what they've learned to real world projects. The chord detection lab from Chapter 2 will teach students about the frequency content of music and how examining signals in the frequency domain can reveal previously unknown characteristics. The musical fingerprinting lab from Chapter 3 will teach students about the powerful combination of newly learned signal processing techniques with computational tools. The JPEG compression lab from Chapter 4 will teach students about the relationship between the spacial and frequency domains and allow them to explore different filters and transforms. The motion magnification lab from Chapter 5 teaches students the relationship between translation in the spacial domain and phase in the frequency domain, as well as filtering techniques in three dimensions.

Bibliography

[1] Cho, T., Weiss, R. J., & Bello, J. P. Exploring Common Variations in State of the Art Chord Recognition Systems. *Music and Audio Research Laboratory (MARL).*

[2] Forney, G. The viterbi algorithm. *Proceedings of the IEEE,* 61(3), 268-278, 1973.

[3] HKN Underground Guide. `https://underground-guide.mit.edu`. Accessed: August 15, 2017.

[4] Kornblum, J. D. Using JPEG quantization tables to identify imagery processed by software. *The International Journal of Digital Forensics & Incident Response*, 2008.

[5] Merrill, M. D. *First Principles of Instruction.* Hoboken, NJ: Wiley, 2012.

[6] MIT Subject Listing and Schedule. `http://catalog.mit.edu/subjects/`. Accessed: August 18, 2017.

[7] Morgan, M. J., & Watt, R. J. Mechanisms of interpolation in human spatial vision [Abstract]. *Nature*, 299(553), 555th ser, 1982.

[8] Mueller, M. *Fundamentals of Music Processing: audio, analysis, algorithms, applications.* Springer, 2016.

[9] Wadhwa, N., Rubinstein, M., Durand, F., & Freeman, W. T. Phase-Based Video Motion Processing. MIT Computer Science and Artificial Intelligence Lab, 2015.

[10] Wang, A. L. (2003). An Industrial-Strength Audio Search Algorithm. Shazam Entertainment, Ltd.