ELEC 301 Projects Fall 2007

Collection Editor:
Rice University ELEC 301

ELEC 301 Projects Fall 2007

Collection Editor:
Rice University ELEC 301

Authors:

Naren Anand
Britt Antley
Kyle Barnhart
Blake Brogdon
Kriti Charan
Janice Chow
Chris Corbet
Akshay Dayal
Thomas Deitch
Mark Eastaway
Taylor Goodhart
Aaron Hallquist
Jason Holden
Tianlai Lu
Niklos Maureira
Tanwee Misra
Alex Mrozack
Justin Nordin
Oleg Pesok
Angela Qian
Jason Ryan
John Steinbauer
CJ Steuernagel
Keith Wilhelm
Thomas Yeh

Online:
< http://cnx.org/content/col10503/1.1/ >

C O N N E X I O N S

Rice University, Houston, Texas

Collection structure revised: December 22, 2007
PDF generated: September 13, 2011
For copyright and attribution information for the modules contained in this collection, see p. 118.

Table of Contents

Chapter 1

Detection of Cell Boundaries in Optical Fiber Probe Images

1.1 Introduction[1]

1.1.1 Introduction

1.1.1.1 Abstract

The nuclear-cytoplasmic ratio (NCR) is a widely used clinical indicator for cancer. The automated extraction of cell size and nuclear size from tissue images is the only practical method to assess quantitatively the NCR in a real-time clinical setting. In this study we demonstrate the feasibility of automatically detecting the cell boundaries and estimating the cell sizes from images of chicken adipose tissue using several image processing techniques and an implementation of the Hough Transform.

1.1.1.2 Goals

Using a completely non-invasive image capturing technique, we want to be able to automate a process to detect the cells in the image, as well as calculate the area of the cells. By doing this, when imaging techniques become more advanced, the process can be used to determine the NCR and thus, become an automated, non-invasive indicator of the presence of cancer.

1.2 Image Acquisition[2]

1.2.1 Image Acquisition

1.2.1.1 The Optical-Fiber System

We have designed an optical system to test the imaging capabilities of a one millimeter diameter optical fiber probe. We used a 455 nm Luxeon LED (Philips, San Jose, CA) as the illumination source in accordance with our design goal of a device capable of both reflectance and fluorescence imaging while remaining as simple as possible. A broadband light source would be more ideal for reflectance imaging. However, a broadband light source could perform fluorescence imaging whereas our source would be suitable for both fluorescence and reflectance imaging. The LED was cooled by a heat sink and fan assembly mounted on its back and powered by a 12V AC/DC adaptor. To mimic the fatty environment of the human breast, test images were taken from samples of chicken adipose tissue.

[1] This content is available online at <http://cnx.org/content/m15686/1.1/>.
[2] This content is available online at <http://cnx.org/content/m15687/1.1/>.

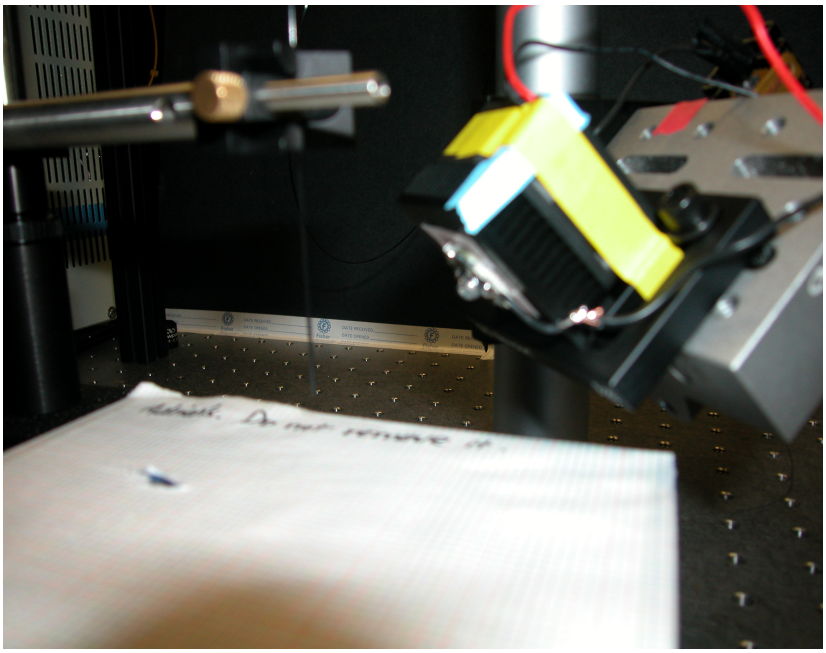


Figure 1.1

The 1-mm diameter fiber (Sumitomo Electric USA, Los Angeles, CA) was positioned next to the LED and over a vertically adjustable sample stage. (Figure 2) The fiber tip was adjusted to receive maximum illumination from the LED when it was in contact with the sample. The fiber collected the light reflected by the sample at its distal end and conveyed it to its proximal end. The proximal end of the fiber was focused at the focal plane of an infinity corrected microscope objective (Newport, Irvine, CA). The position of the fiber was adjusted by a three-axis micro translation stage (Newport). The infinity corrected object could project an image from its rear that is in focus regardless of the distance between the object and the detector.

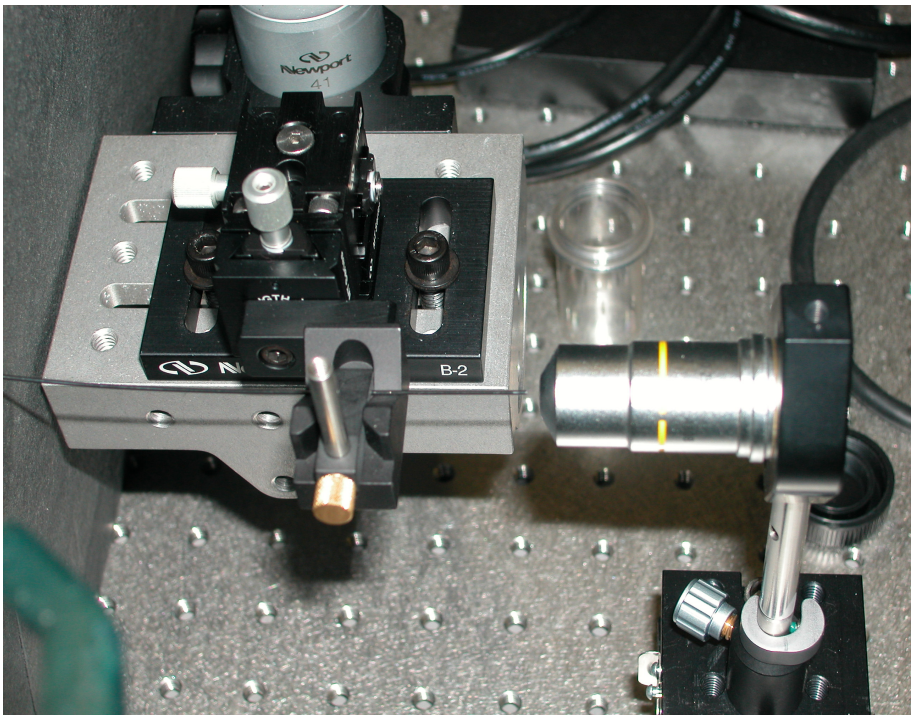


Figure 1.2

The detector was a Zeiss MRc5 5.0 megapixels color CCD camera (Carl Zeiss, Thornwood, NY). The magnification of the image on the detector relative to the image on the proximal fiber tip was the ratio between the distance from the detector to the back of the objective and the distance from the front of the objective to the fiber proximal tip. The infinity corrected objective permitted a wide range of magnifications (a while range of allowable detector-objective separation) while maintaining the focus of the image. (Figure 3) This feature could potentially allow the incorporation of a movable detector that can change the magnification of the image on command.

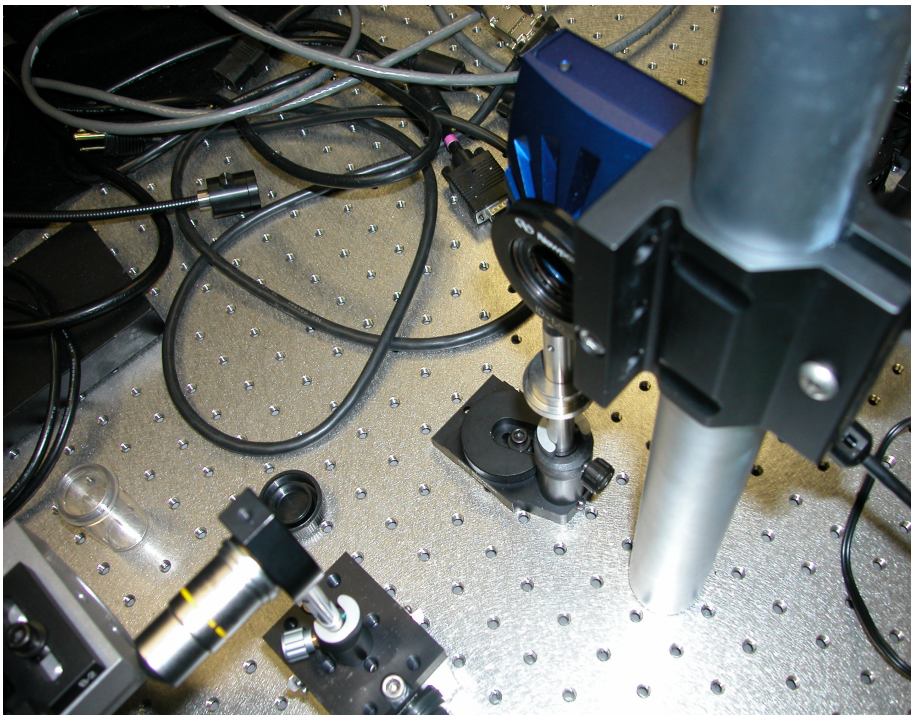


Figure 1.3

The image from the detector was captured onto a laptop computer (Lenovo, China) with the Zeiss AxioVision camera software. (Figure 4) The image was exported as a bitmap file. The bitmap file was subsequently processed in Matlab (Mathworks, Natick, MA).

Figure 1.4

1.3 Defining Borders[3]

1.3.1 Defining Borders

1.3.1.1 Overview

With the original cell picture, as shown below, it is very difficult to make out the cell borders due to the lack of contrast between the borders and the background of the image. Our first goal, before edge detection can even be done, is to create a sharper contrast between the borders and the background. This is shown in Figure 1.

[3]This content is available online at <http://cnx.org/content/m15688/1.1/>.

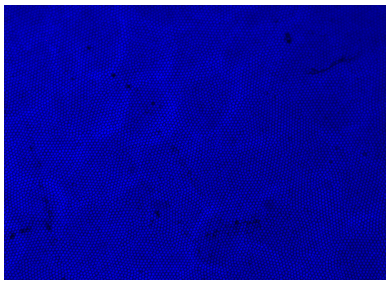


Figure 1.5

1.3.1.2 Thresholding

In order to create a higher contrast between a cell's borders and the background, thresholding was utilized. Pixels of lower intensities, the background, were thrown out while pixels of higher intensities were kept. The IMADJUST command in Matlab allowed us to specify a threshold intensity and mapped all pixels below the threshold to zero. The result can be seen in Figure 2.

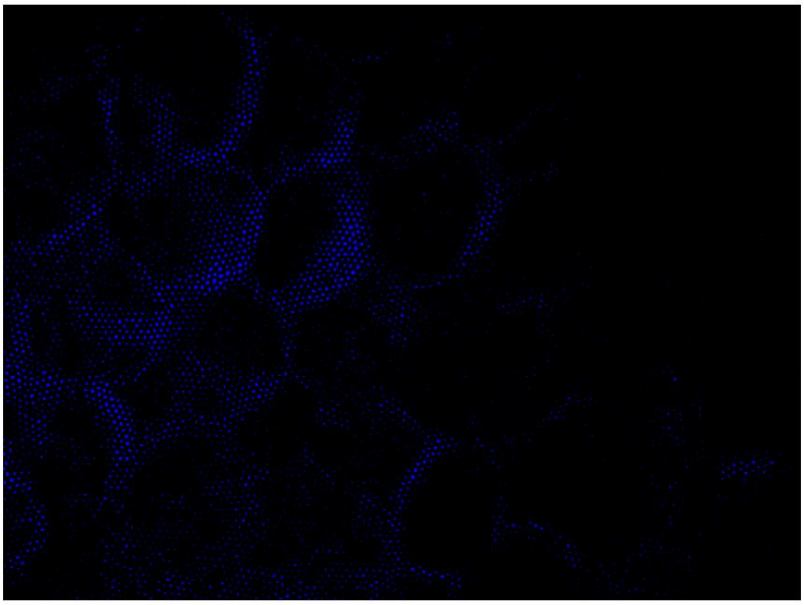


Figure 1.6

1.4 Filtering[4]

1.4.1 Filtering

1.4.1.1 Overview

Although we have more contrast between our cell borders and the background, there is still a lot of noise which will need to be annihilated before edge detection can be successful. Ideally, we want to only preserve pixels which constitute a cell border and map all other pixels to a zero intensity.

1.4.1.2 Conversion to Gray Scale

Filtering is more easily done on a grayscale image than on a colormap image. This is due to the fact that grayscale images are intensity images with each pixel reflecting a particular intensity, rather than a combination of three colors (red, green and blue). With a grayscale image, filters such as mean and mode filters can be more appropriately applied.

1.4.1.3 Mean Filtering

The first step in getting rid of unwanted pixels is to apply a mean filter. This filter will take a specified neighborhood around each pixel and set each pixel to the mean of its neighborhood. This is done with Matlab's sliding neighborhood operation which will go through each pixel, taking its neighborhood pixels as inputs into a specified function and set each pixel as the output of the function.

In our case, the function we want to apply will take the neighboring pixels and output the mean. This mean represents the average intensity of its neighboring pixels. Thus, a lone intense (lit) pixel way out in the boondocks will be set to zero since there are no other high-intensity pixels nearby. This will effectively eliminate lone high-intensity pixels and thus decrease noise.

Following mean filtering, the image was converted to black and white. This was done by specifying a threshold intensity and setting all pixels above the threshold to 1 (white) and all pixels below to 0 (black). Why must this be done? Well, you'll just have to read on to see... Figure 1 shows our final result in this step after converting to black and white.

[4]This content is available online at <http://cnx.org/content/m15689/1.1/>.

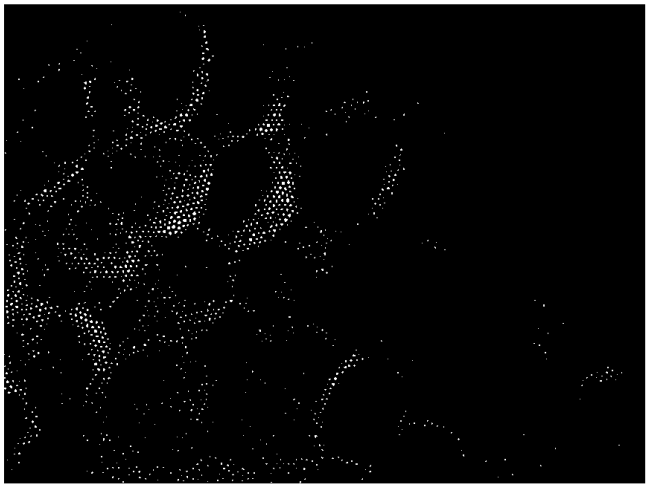


Figure 1.7

1.5 Connecting Lines[5]

1.5.1 Connecting the Lines

1.5.1.1 Rationale

OK, so now we have a (relatively) clean image of the cell's borders. Can we run an edge detector now you ask...? Hold your horses, Champ, we still have a ways to go...

Although it may make sense to run an edge detector at this point, seeing as we've got ourselves an image where the cell borders are definitely distinguishable, there is still a problem. Edge detectors look for changes in the gradient. Since our image was acquired utilizing optical fibers, our cell borders are not really solid lines quite yet; they are merely groups of small dots which, together, make up the cell borders. If an edge detector were utilized at this point, it would pick up each fiber optic probe, rather than the cell border we want. There no need to despair; there is a solution for all this!

1.5.1.2 Mode Filtering

Although it may not be as simple as those connect-the-dots books we're used to, mode filtering can be an effective method of forming solid cell borders. The basic idea is this: if we used sliding neighborhood operations again, but rather than look for the mean, look for the mode, we may be able to connect all our little dots together! Why is this? Well, the mode filter we implemented works like this:

1. Take a neighborhood of size [N M]
2. Find the mean of the entries in the matrix.
3. Since the image is in black and white, if the mean is greater than .5, then there are more ones, otherwise, there are more zeros. This is basically determining the mode of the neighborhood.

[5]This content is available online at <http://cnx.org/content/m15690/1.1/>.

4. Thus, if the mode is 1, set the pixel to 1. If the mode is 0, set the pixel to 0.

How does this connect the lines? Think about this: we want to establish solid cell borders by turning ON the black pixels which are part of the cell's border. Thus, we want to turn ON black pixels which are near groups of white pixels which make up the cell borders and keep black pixels which are not part of the cell borders (in areas with relatively few white pixels) OFF. Taking the mode of each pixel's neighborhood will accomplish this since black pixels near large groups of white pixels will be turned on and the following image shows what begins to look like solid cell borders... (Figure 1)

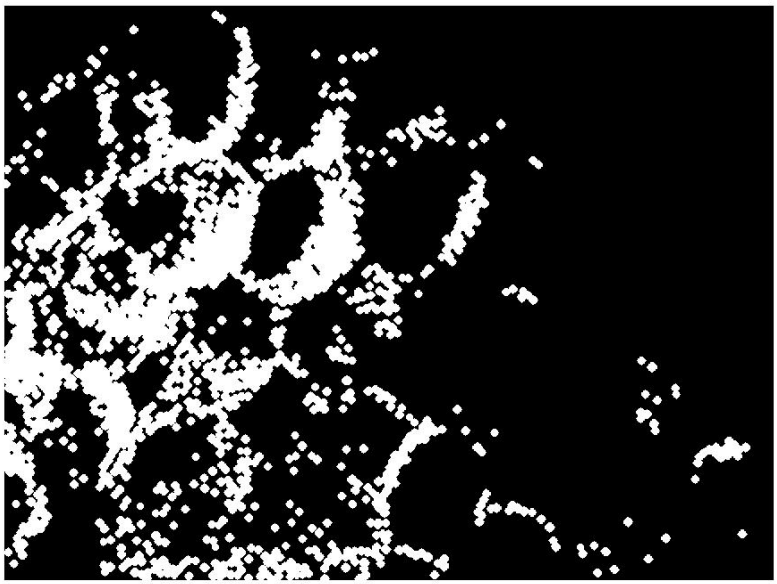


Figure 1.8

1.6 Edge Detection[6]

1.6.1 Edge Detection

1.6.1.1 Canny Edge Detector

We use the Canny Edge Detector to (hopefully) obtain sharp edges for our image now that we have "blurred" together the dots which form the cell edges. The Canny edge detector first takes a gradient of the image. A gradient magnitude, as well as the edge direction (which way the gradient is increasing) can then be found. The Canny edge detector does this by using four different filters which can detect horizontal, vertical and diagonal edges. For each pixel, the filter which outputs the largest response can be found and the direction determined.

[6] This content is available online at <http://cnx.org/content/m15691/1.1/>.

1.6.1.2 Distance Matrix

Once we have run our image through the Canny edge detector, a distance filter is used to connect all remaining lines in a cell's border. The distance function calculates the Euclidean distance between each pixel and the nearest pixel which is on. The output of the distance function is a matrix having the same dimensions as the image matrix. The entries correspond to the distances; for example, if pixel (i,j) was a distance x away from the nearest on pixel, the (i,j) entry in the distance matrix will be x.

After the distance matrix is found, an iterative method is used to go through each entry of the matrix. A threshold distance is determined and every entry of the distance matrix which falls under the threshold is mapped to one, and every entry which falls above is mapped to zero. What exactly is this doing? Well, if a pixel is close enough to an on pixel, then we assume it is part of the cell's border, and thus must be turned on as well. The distance function and subsequent thresholding is performed multiple times, with the threshold getting small and small each time until the cell borders are completely continuous. The result is shown below.

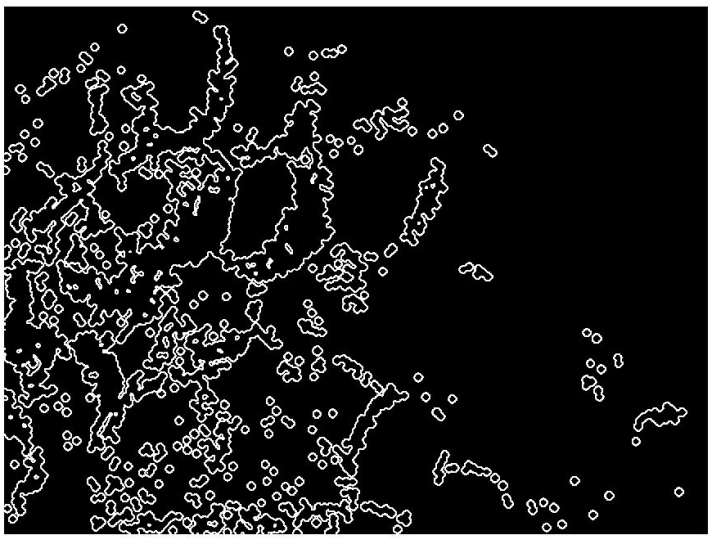


Figure 1.9

1.7 Hough Transform[7]

1.7.1 And now, Introducing the Hough Transform...

We are finally about to get to the fun part: circle detection...

1.7.1.1 The Hough Transform

The Hough Transform is performed to recognize circular patterns based on parameters such as circle radius range, gradient threshold and presence of concentric circles. The Hough Transform will in as an input a

[7] This content is available online at <http://cnx.org/content/m15692/1.1/>.

range of radii. It will then generate circles of varying radii which cover the entire range. These circles are convolved with the image and the goodness of match is recorded in an accumulation array.

Note: Generally, convolution is not performed, but rather, the FFTs are taken and multiplied together. This is not only more efficient, but more easily implemented as well!

We found a nifty Matlab function which implements the transform and ran it on our image. The accumulation graph can be seen below, with the peaks corresponding to the best circle matches.

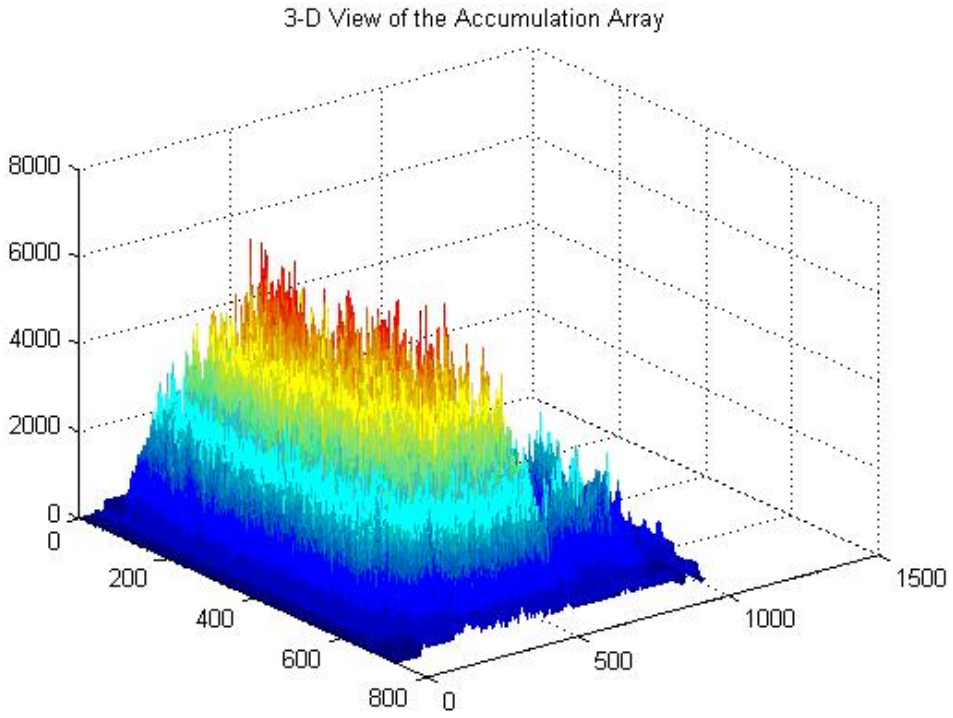


Figure 1.10

1.7.1.2 Circles, Circles, Circles!

With the circles detected, the areas where the best circular matches were can be drawn out in our original image. The result is quite striking: of the twelve or so cells which could be picked out by the human eye, the Hough Transform was able to detect roughly eight cells.

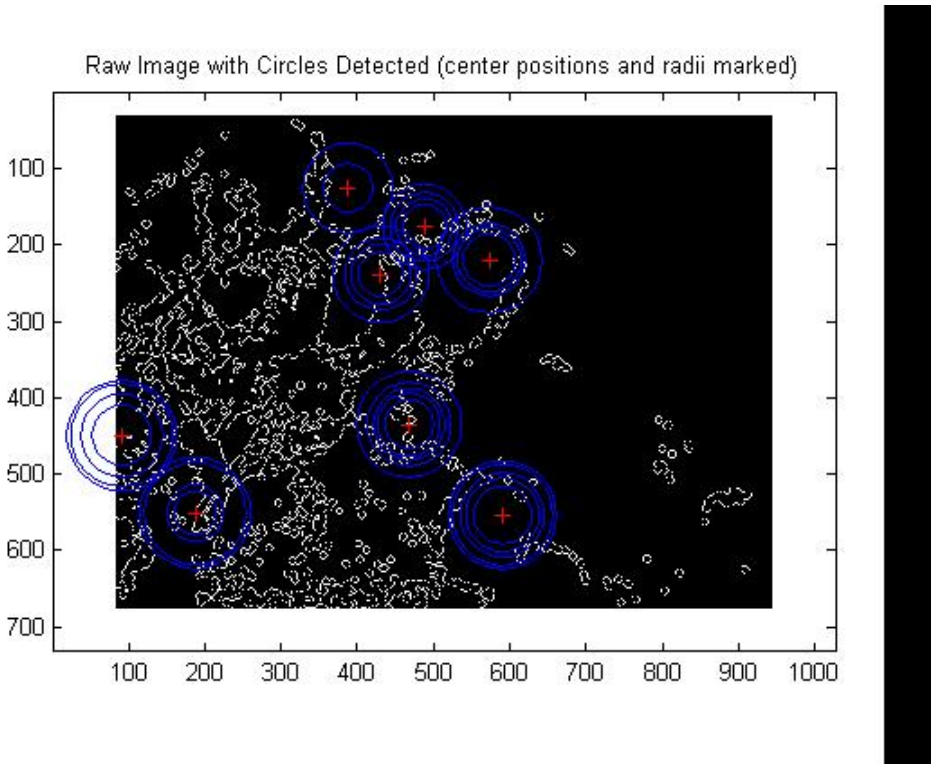


Figure 1.11

1.8 Conclusion[8]

1.8.1 Conclusion

1.8.1.1 Area Calculation

With the accumulation array, we were able to extrapolate the sizes of the individual cells and it was determined that the average cell size in our image is roughly 150um2.

1.8.1.1.1 But what does it all mean?

When imaging techniques become more advanced, it is our hope that not only would we be able to calculate the size of the cells, but the nucleus as well. By being able to calculate both the cell sizes as well as the nucleus, the ratio between the two can be determined. This can then be used as a completely non-invasive detector of cancer, with higher nucleus-to-cell ratios being indicative of the presence of cancer.

1.8.1.2 Improvements

Our area calculation is far from perfect and, with time, there are many improvements which can be made.

[8] This content is available online at <http://cnx.org/content/m15693/1.1/>.

1.8.1.2.1 Image Acquisition

The image acquisition technique can be improved to not only be able to detect the nucleus, but to provide better definition between a cell's border and the background as well. But... this we will leave to our bioengineering friends.

1.8.1.2.2 Edge Detection

In our edge detection phase, one problem we faced was the varying thickness of cell borders. Sometimes, especially thick borders would appear as a closed region resembling a cell when edge detection is performed! This problem, however, we believe can be rectified using carefully constructed filters which can hopefully reduce the thickness of borders.

1.8.1.2.3 Circle Detection

The Hough Transform we used approximates cell sizes using the approximate circles. Cells, however, are not always completely circular. A more generalized transform can be employed to more precisely detect the shape of the cells and give a more precise area calculation.

1.9 Team[9]

1.9.1 Project Group Members

1.9.1.1 Chenghao Shu, Bioengineer

Figure 1.12

[9] This content is available online at <http://cnx.org/content/m15694/1.3/>.

1.9.1.2 Demetrius Austin, Electrical Engineer

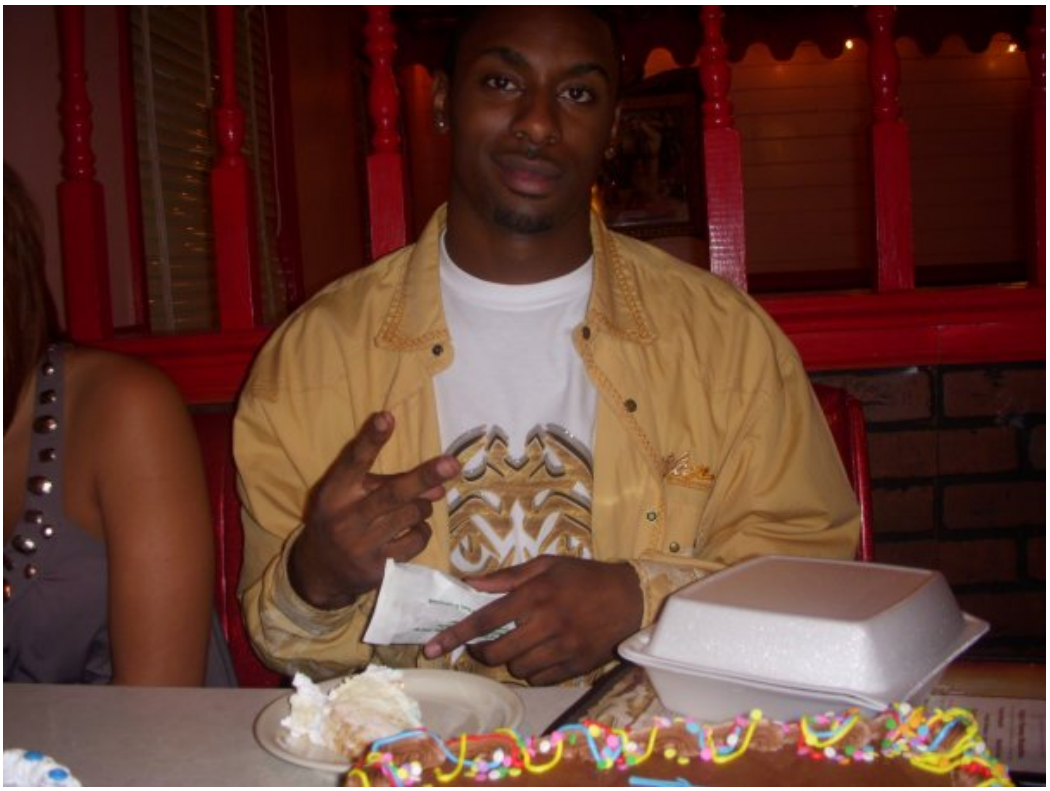


Figure 1.13

1.9.1.3 Thomas Yeh, Electrical Engineer

Figure 1.14

1.9.1.4 Chirag Sequeira, Electrical Engineer

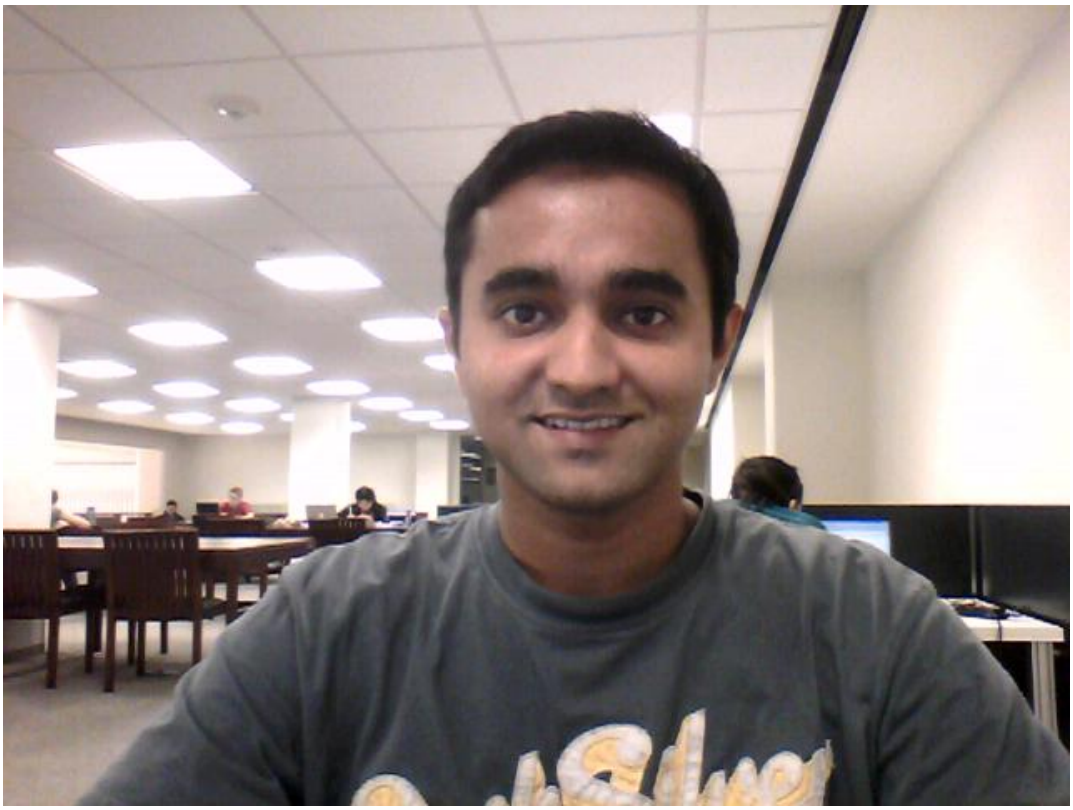


Figure 1.15

Chapter 2

Tap That: A Virtual Drum Kit

2.1 Introduction to the Virtual Drum Kit[1]

Imagine This:

Having your own invisible and portable drum kit that not only takes up far less space, but also costs ten times less than a real one. With virtual reality no longer a distant dream and technology headed towards its implementation in all applications, a virtual interface for musical instruments is inevitable. This is the focus of "Tap That" – creating a cost-effective, compact and enjoyable drumming experience by means of a virtual drum kit.

2.1.1 Past Work

The first virtual drum kit was created by a French group (www.virtual-drums.com[2]). Their implementation uses two webcams for 3D spatial analysis and extensive C++ coding. There are several other virtual drum kits available on the internet, but they must be controlled with either a mouse or a keyboard, which detracts from the drumming experience. The drum kits developed by the French group and our group give the user a real feeling of playing the drums while actually being virtual.

2.1.2 Our Goal

This project aims at creating an affordable virtual drum kit for amateur drummers using readily available materials. Music enthusiasts can "Tap That" with this cost-effective replacement of a real drum kit. This instrument requires a web camera, Matlab 6.0 or higher, two LED lights (mounted on drumsticks for a more authentic experience) and a computer.

[1] This content is available online at <http://cnx.org/content/m15698/1.1/>.
[2] http://www.virtual-drums.com/

2.1.3 Summary

This project is based on a 2-dimensional tracking system. It tracks the LEDs in two dimensions and uses velocity computations to determine when and where the drum was hit. The user positions himself or herself on a chair in front of the webcam and runs the program. An animated image on the screen gives the user the approximate positions of the drums. As the user moves the LEDs in the air, pretending to hit real drums, Matlab calculates the position of the LEDs by carrying out a frame-by-frame analysis of the video in real-time. The velocities of the LEDs are calculated, and when the LED starts moving upwards instead of downwards, it means that a drum has been hit. Based on the position of the strike, the corresponding drum's sound will be produced while the animation gives the user feedback about which drum they hit.

2.2 A General Approach to Building Your Own Virtual Drum Kit[3]

2.2.1 Flow Chart

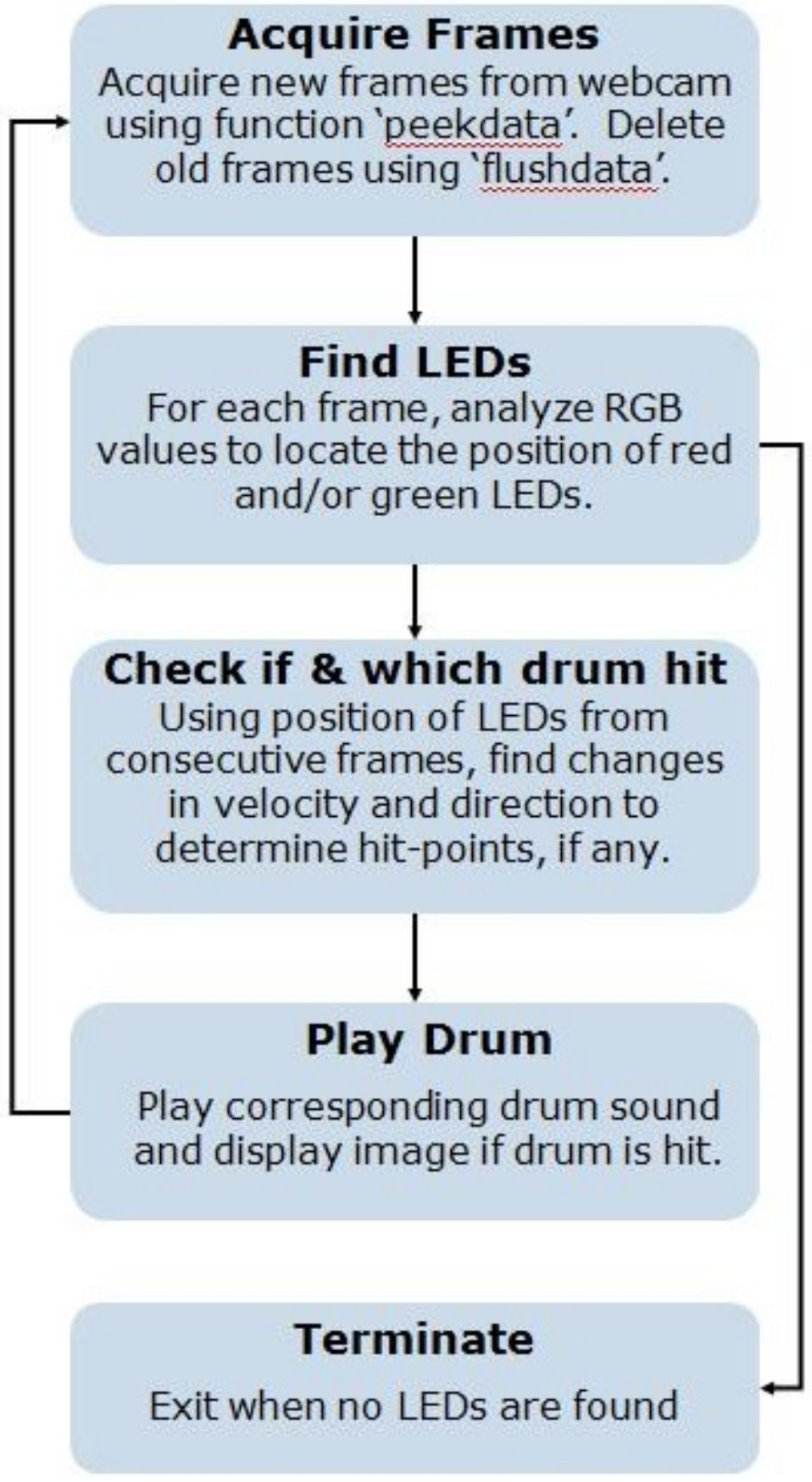


Figure 2.1

2.2.2 The LED Drum Sticks

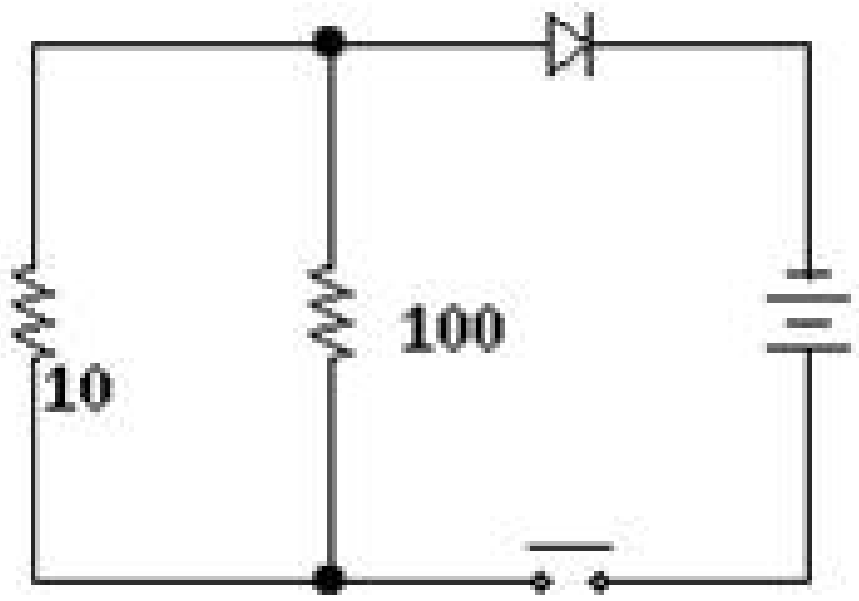


Figure 2.2

The LED drumsticks consist of a simple LED circuit mounted on narrow pipes. Instead of using a battery holder and a switch, we attached a pen flashlight to the circuit. The program also requires that one of the LEDs be red, and the other green. The circuit consisted of a super-bright LED, 9.11 ohm of resistance and a 3 V source. A lower resistance will make the LED brighter, making its detection better. Alternatively, the user can obtain LED sticks on the internet for $5 each. When the switch at the end of the flashlight is turned on, the LEDs will light up and serve as detection points of the drumsticks. The program stops running if you switch off the LED drum sticks.

The only limitation posed by the LED drumsticks is that they have to be used in a dimly lit area to be detected without errors.

Now that you have your drumsticks, you are all set on the hardware and need to implement the software!

Figure 2.3

2.3 Webcam Color Tracking in Matlab[4]

2.3.1 Back ground: Image Processing in Matlab

2.3.1.1 What composes an image?

Each image is composed of an array of M*N pixels (contraction of "picture element") with M rows and N columns of pixels. Each pixel contains a certain value for red, green and blue. Varying these values for red, green, blue (RGB) we can get almost any color.

[4]This content is available online at <http://cnx.org/content/m15696/1.2/>.

Image Storage in Matlab

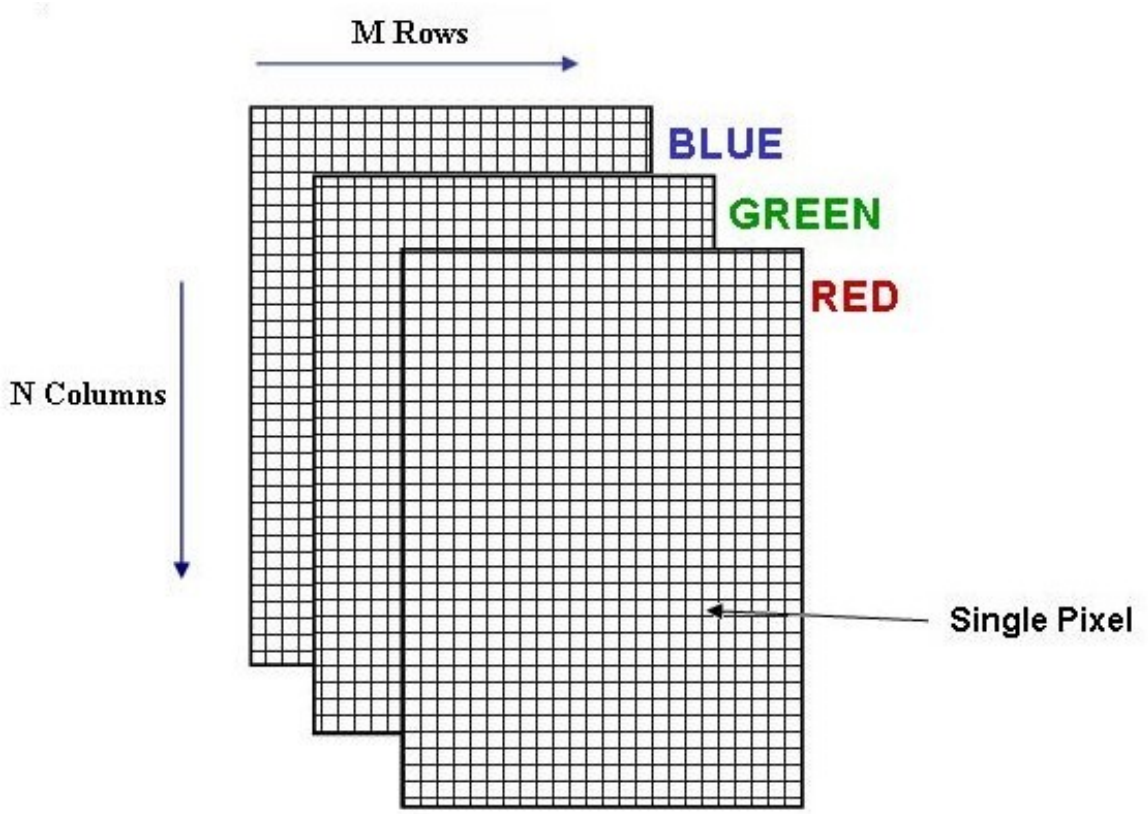


Figure 2.4

2.3.1.2 Color Detection

The RGB format is a practical method to represent color images. Matlab creates three matrices (or three M x N arrays) with each matrix representing normalized components of red, green or blue to read and store each of the frames of the video. Any pixel's color is determined by the combination of Red, Green and Blue values stored in the three matrices at that pixel's location. This is how Matlab reads and manipulates .jpg files.

Images:

`picture (row, column, rgb value)`

For example, picture (12, 78, 1) corresponds to the red component of the pixel at row 12 and column 78; picture(12, 78, 2) corresponds to the green component of the pixel and picture(12, 78, 3) gives us the blue component of the pixel at that location.

Videos:

`frames(row, column, rgb value, frame)`

Videos have an extra dimension for the frame number. So frames(12,78,1,5) would correspond to the red

component of the pixel in the 12th row and 78th column of the 5th frame. To get the entire frame, we could just say frames(:,:,:,5).

2.3.2 Acquiring Images From The Webcam in Matlab

Digitization of the Images

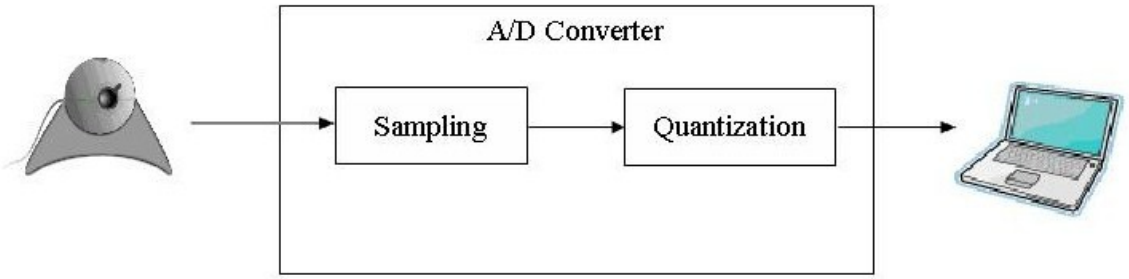


Figure 2.5

How you acquire the images from the camera depends a lot on what software you are using to implement it. Creating the interface between the computer and the drum using a lower level language like C or C++ will give you a lot of flexibility, but it will also involve a lot of work and background knowledge. Fortunately for us, Matlab's Image Acquisition Toolbox has a variety of simple functions that can be used to create the interface. The next few paragraphs will describe these functions and how we used them in some detail.

For Matlab to recognize the video camera you have to create it as an object using the command `obj=videoinput('winvideo')`. Matlab will automatically find the webcam connected to your computer. Once it is an object in your workspace you can edit its settings, such as the number of frames per second, to optimize it for your project. `preview()` and `getframe()` are two useful functions for determining if the camera has been positioned properly. The first allows you to see what the camera sees, without collecting any data from it, and the second acquires a single snapshot and stores it as in image.

Now we can start recording the video. With the `start(obj)` command, the camera will be triggered and will start to collect as many frames as specified by the **FramesPerTrigger** property. These frames are stored in the camera's buffer memory. These frames are stored in the 4D array as described above.

To retrieve them from the buffer you could use either the `getdata()` function or the `peekdata()` function.

2.3.2.1 getdata(obj);

When this statement is encountered, the program retrieves all the frames acquired at the last trigger and then empties the buffer.

2.3.2.2 peekdata(obj,M);

This function allows us to "peek" at the last M frames collected by the camera. The frames are copied, but not cleared, from the camera's buffer into Matlab's workspace.

Both functions take about the same amount of time to run. Interestingly, the number of frames acquired at a time does not affect the execution time much. It takes about as long to acquire 1 frame as it does to acquire 50. Therefore, to make the program more efficient, we should collect large chunks of data at a time.

To make sure that we don't miss any of the action while the user is waving the drumsticks around in front of the camera, we should also make sure that we can get all the frames from when the program starts running till the user turns of the LEDs. Due to memory limitations, our computer could collect maximum of 240 frames, or about 8 seconds of the video, which is clearly not enough. It is unlikely that your computer could do much better.

2.4 Implementation: Detecting a Hit[5]

2.4.1 Hitting the Drums: A Velocity Computational Approach

You now have the positions of the LEDs in every frame. Our goal here is to detect when and which drums were hit so that the corresponding drums can produce their sounds.

2.4.1.1 Determining when a drum was hit

There are various innovative ways of tracking when the drums are hit. We chose to use a velocity computational approach due to its speed. Using this approach, you can look at the displacement of the LEDs in consecutive frames and calculate the velocity. Using basic physics, if the velocity changes from positive to negative (note that our coordinate system has the positive side of the y-axis pointing downwards) it implies a change in direction of the drumstick, or that a drum was hit. You should be able to figure out approximately which frame the drum was hit in. Since the hit point is returned in terms of two-dimensional coordinates, you can figure out which drum(s) are hit.

[5]This content is available online at <http://cnx.org/content/m15714/1.1/>.

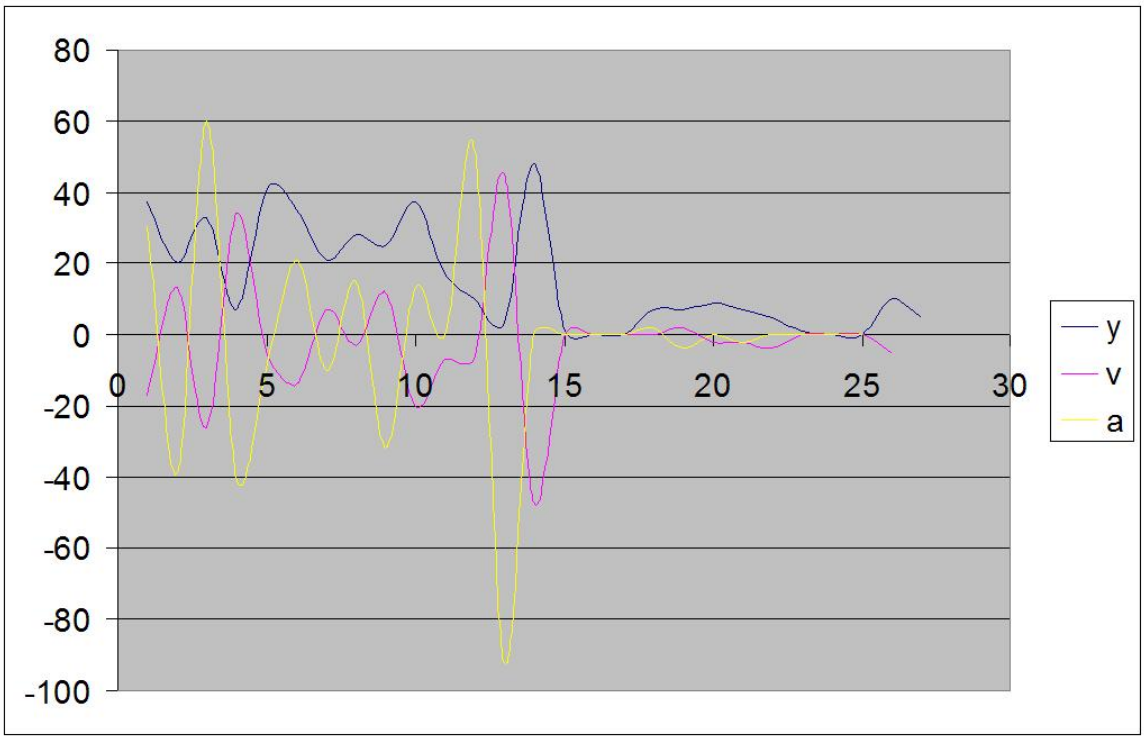


Figure 2.6: See if you can guess when drums were hit.

2.4.1.2 Determining which drum was hit

Position of the Drums

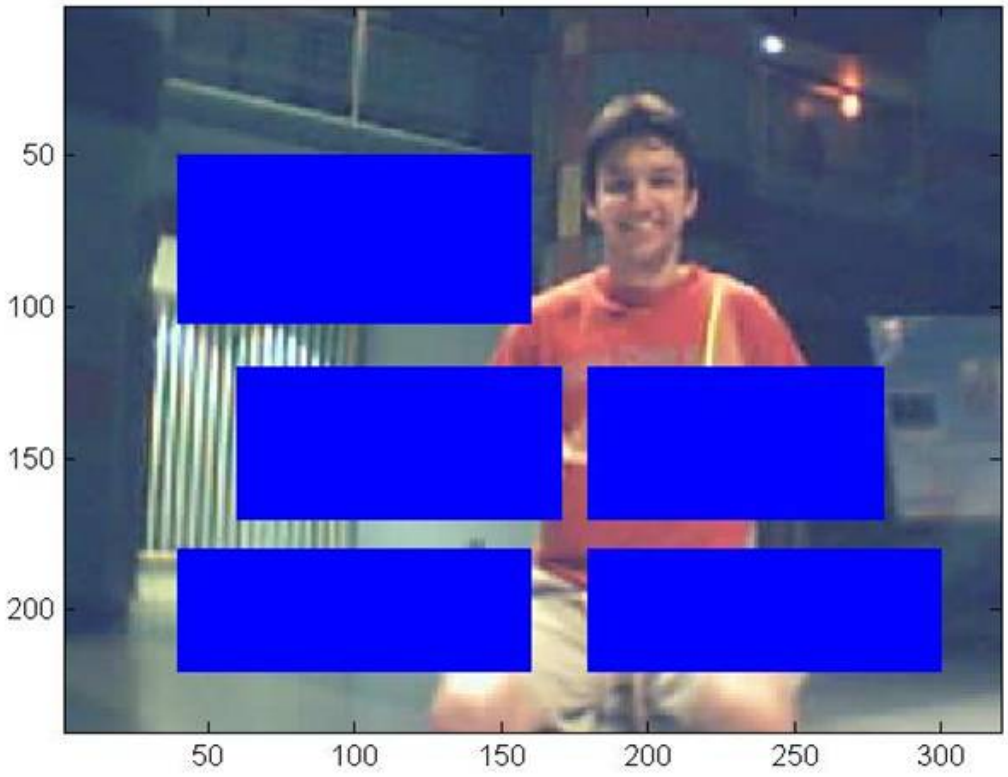


Figure 2.7: [1: Cymbal; 2: BigTom; 3: Small Tom; 4: Floor Tom; 4: Snare]

The frame should be broken down into rectangles with assigned x and y coordinates where the drums are positioned. Our drum kit consists of the Crash Cymbal, the Big Tom, the Small Tom, the Floor Tom and the Snare. We check to see if the hit-point falls in the space assigned for any of the drums, and if it does, we move on to producing the sound and displaying the image on the computer screen. However, if the hit point is outside the frame, it is taken as a missed hit.

2.5 Implementation: Detecting the LEDs[6]

2.5.1 RGB in Each Individual Frame

Given the acquired frames, the first step is to locate the LEDs. The color of the LEDs can be described by the intensity of red, green and blue. These are represented by values between 0 and 255 (going from null intensity to full intensity) in the RGB matrices for each image. We can detect the LEDs by setting threshold values for Red, Green and Blue. For example, to find the Red LED, you are looking for the part of the frame with high Red values and low Blue and Green values. We would recommend using the data cursor to analyze RGB values of the LED in the colormap. Play around with threshold values until you get the right thresholds, i.e., only the LED you are trying to detect shows up.

[6]This content is available online at <http://cnx.org/content/m15699/1.1/>.

A Sample Frame

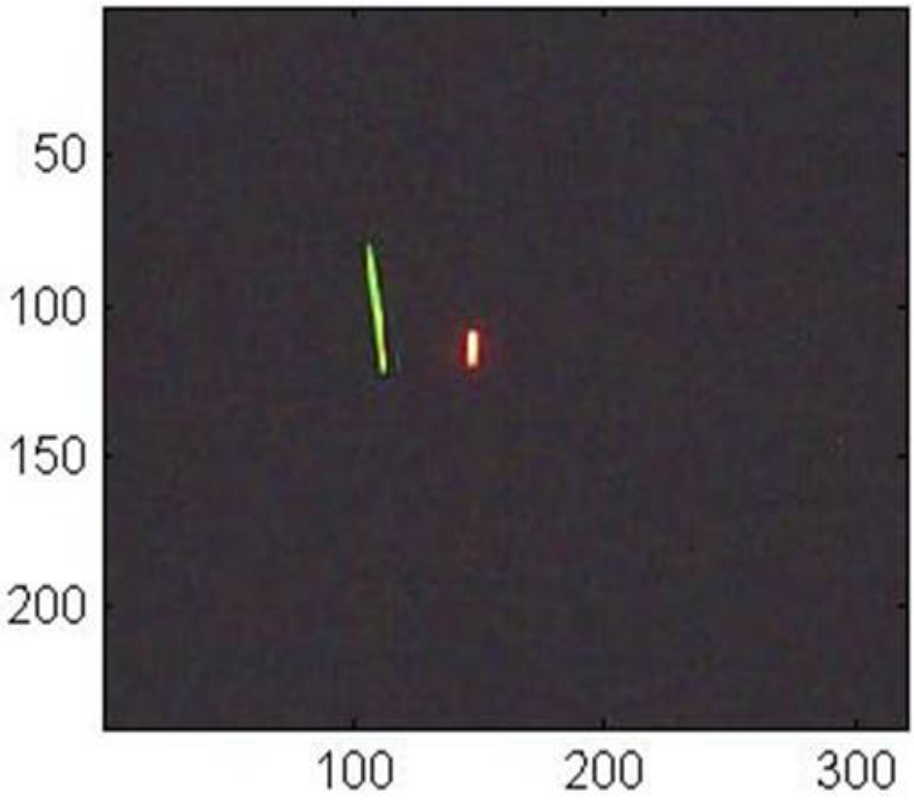
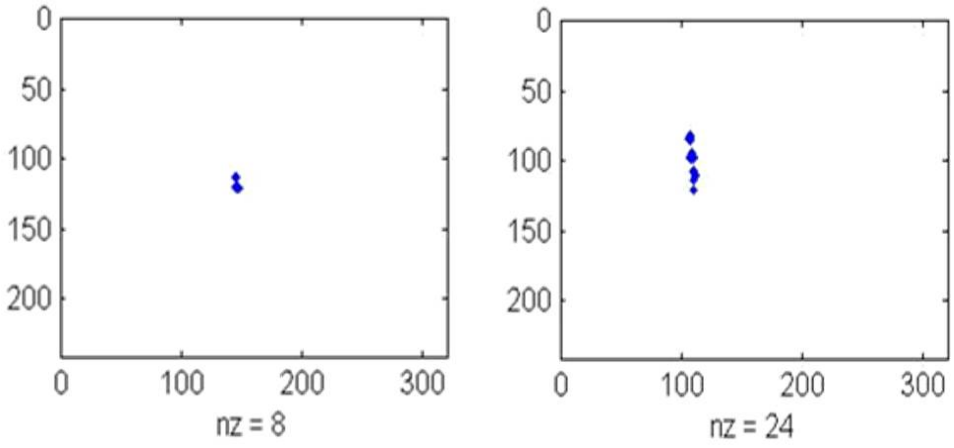


Figure 2.8

Spy Plots of the LEDs



(a) (b)

Figure 2.9: (a) Red LED (b) Green LED

In our project, we used the find function to look at the red, green and blue matrices separately and set ones wherever the conditions were satisfied. We multiply the three matrices of ones and zeros to produce one final matrix with ones where all three color thresholds are satisfied and the LED is detected.

2.5.2 Turn Off the LEDs

If the LEDs are not detected for 75% of the frames, it means that the LEDs are either turned off or not in the frame at all, and the program stops running. However, if the LEDs have been found, you can move on to detecting when the drums are hit.

2.6 Producing Drum Sounds and Displaying the Drum[7]

2.6.1 Producing Drum Sounds and Displaying the Drum

Once we have determined which drum has been hit, we play the appropriate sound and display the corresponding image. The drum sounds and images are preloaded in separate .mat files at the beginning of the program using the load command to expedite execution.

```
load drumsounds
load drumpics
```

2.6.1.1 Drum Sounds:

`drumpics.mat` was created by importing drum samples in the wav format using `wavread()` and `wavadd()`. `wavread('wavfile')` is a Matlab function that stores wav files as an Nx2 matrix. `wavadd(sound1,sound2)` is a function that we created to allow the simultaneous play of two different sounds. For example, if both the snare and the crash cymbal were hit at the same time, we zero pad the shorter matrix and combine them as

[7] This content is available online at <http://cnx.org/content/m15697/1.1/>.

one sound matrix. In the main program, the Matlab function `soundsc(sound, fs)` plays the corresponding sound file, where 'fs' is the sampling rate. Our wav files were sampled at 22 kHz to allow for smaller sound variables without affecting the sound quality too much. All of our drum samples were obtained free online from a Yamaha 9000 drum kit.

2.6.1.2 Drum Pictures:

`drumsounds.mat` was created by importing jpeg files using the `imread()` Matlab command to store the images as variables. The images were created from Google SketchUp, a free 3D modeling program available online. Different views were created to simulate the feel of animation.

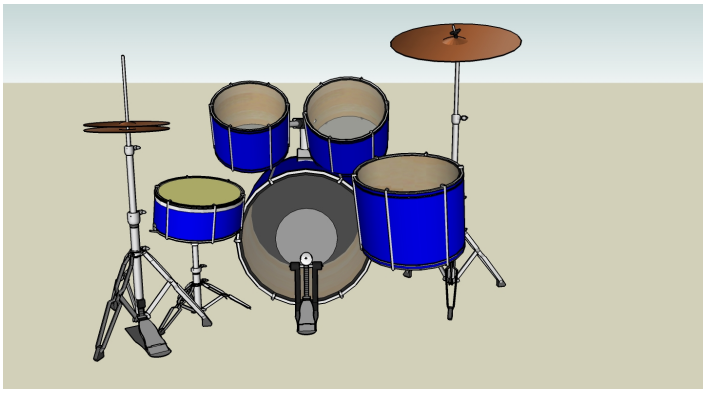


Figure 2.10: Main view

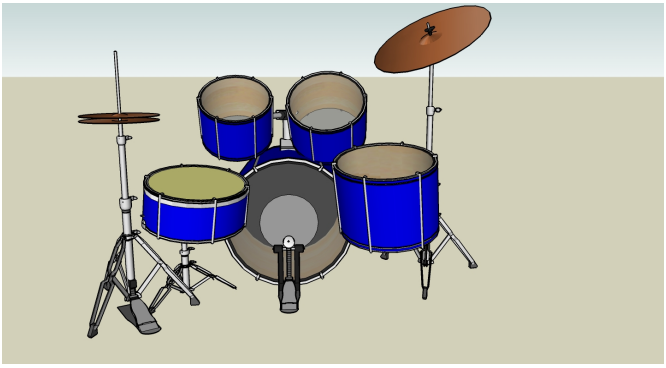


Figure 2.11: Crash cymbal and snare were hit

2.7 Implementation: Code[8]

To get a better understanding for how we implemented the drum kit, you might want to have a look at our Matlab code.

The code is available at: http://www.geocities.com/tapthat_virtualdrum/tapthat.html

2.7.1 Essential Files:

proj_301.m- This is a wrapper function to set up video acquisition and collect frames from the camera.

detectcolor.m- This function checks for presence of the LED.

coord.m- This function finds the position of the LED in each frame, computes its velocity and hit-point.

drumdet.m- This function determines which drum was hit.

drumdisp.m- It produces the sound and displays the image corresponding to the drum that was hit.

2.7.2 Files for debugging and setting up the system:

adjust.m- This is mainly for the user's convenience. It helps adjust the position of the user with respect to the drums.

checkframe.m- This can be used to verify that the color detection works properly for a particular frame.

2.8 Results of the Virtual Drum Kit[9]

2.8.1 Results

While there are many ways to approach the virtual drum kit, we are pleased with the outcome of our implementation using one webcam and velocity computation. We found that users enjoyed the drum kit more after familiarizing themselves with the set-up.

2.8.1.1 Advantages

- High precision in detecting where the drums are hit. Our results showed that less than two hits were missed in a minute's worth of playing.
- Affordable solution for drumming enthusiasts.
- Easily Transportable.
- Runs in real time without a significant delay with Matlab set to high priority.
- Easy to modify the Matlab code to make the system customizable.
- Animated interface gives a visual representation of drums as feedback hit instead of just bare audio.
- Absence of physical drums makes the drummer visible to the audience.

2.8.1.2 Limitations

- Slight delay due to processing power of the computer.
- Requires a dimly lit area for easy LED detection.

[8] This content is available online at <http://cnx.org/content/m15704/1.1/>.

[9] This content is available online at <http://cnx.org/content/m15711/1.1/>.

2.9 Future Work for the Virtual Drum Kit[10]

2.9.1 Conclusion

While the virtual drum kit is not advanced enough to replace a real drum kit for a professional drummer, it is an enjoyable alternative for anyone aspiring to learn how to play the drums or amateur music enthusiasts looking for an affordable solution.

Further improvements in our implementation could lead to a drum kit that is almost as good as a real drum kit.

2.9.1.1 Room for Improvement

- Addition of the bass drum, hi hat, ride cymbals for a fuller drum kit.
- Incorporation of a change in volume based on the acceleration with which drums are hit.
- Higher frame rate and more powerful machines to overcome the delay and visual latency.
- Adding customizability to allow the user to modify the positions of the drums, and the sounds that are played.

2.9.2 Future Work

The concept of LED tracking forms the base of many useful implementations. More specifically, our Matlab code can be easily modified to implement a virtual whiteboard or any other percussion instrument.

With a sense of adventure and some degree of technological innovation and creativity, virtual musical instruments can replace expensive physical musical instruments while delivering comparable results.

2.10 Team Members and Acknowledgments[11]

"Tap That" was brought to you by Janice Chow, Tanwee Misra and Kriti Charan.

Kriti Charan maintained the caffeinated group's sanity and ensured that we stuck to the cynical 6-point project plan as outlined in Mr. Dye's office. She connected the webcam to the computer. Not just by plugging the USB cord in, but by writing proj_301.m. She also channeled her fear of insects into debugging the code. Basically she was the project guru.

Janice Chow supplied us with free coffee and was responsible for deciding where the user had to hit to make the program react in a certain way, and what those reactions should be. In other words she wrote drumdisp.m and drumdet.m, and by nefarious means, obtained all the images and sounds that were required. She was also a great test subject to see if our program was working or not and tremendously enjoyed playing with Google's ketchup.

Tanwee Misra kept us entertained with her French and was responsible for the program's ability to see color and use that to figure out what the test subject was doing with the drumsticks. In other words she wrote coord.m and detectcolor.m. She also encouraged group projects by working with Janice to make the poster, and Kriti to make the drumsticks.

We are grateful to **Mr. Michael Dye** and **Dr. James Young** for all the help they provided as we made and repaired the drumsticks. We are also grateful to **Dr. Kronister** for unknowingly providing us with so much.

We would also like to thank **Dr. B (aka RichB)** and **Mr. D (aka Mark Davenport)** for all the help, support and guidance that they have provided at every step along the way.

Special thanks to all our **rock star fans** who tried our drum kit on the day of the presentation.

[10] This content is available online at <http://cnx.org/content/m15709/1.1/>.
[11] This content is available online at <http://cnx.org/content/m15713/1.1/>.

Chapter 3

Blind Source Separation via ICA

3.1 Blind Source Separation Via ICA: Introduction and Background[1]

3.1.1 Blind Source Separation via ICA

3.1.1.1 Introduction and Background

Imagine you are driving in a car with the radio on and your friends talking in the backseat. You wish to make a phone call on your Blue-Tooth cell phone; however, the person on the other side of the line cannot distinguish what you are saying due to the background noise. A digital signal processing system can be developed to extract your voice signal from the rest of the noise and send it over to the cell phone tower. One such system capable of doing this is blind source separation.

In this project, we present an introduction to the implementation and study of blind source separation through independent component analysis (ICA). The aspiration of our project is to recover independent source signals given only sensor recordings composed of unknown linear combinations of the independent sources. Through ICA, we can successfully separate the two signals apart or extract a signal (i.e. a voice) from background noise (i.e. music).

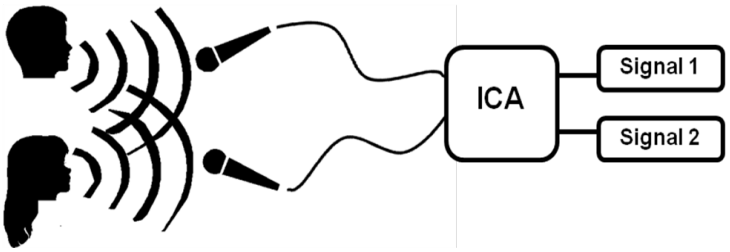


Figure 3.1

Figure 1: This is a pictorial representation of the process described above.

ICA separates the signals using second-order statistics and reduces the higher order statistical dependencies in order to make the recovered separated signals as independent as possible.

Applying blind source separation via this method will enable many applications in signal processing such as audio or image separation, telecommunications, and medical signal processing.

[1] This content is available online at <http://cnx.org/content/m15702/1.1/>.

3.2 Blind Source Separation Via ICA: Implementation[2]

3.2.1 Blind Source Separation via ICA:

3.2.1.1 Implementation

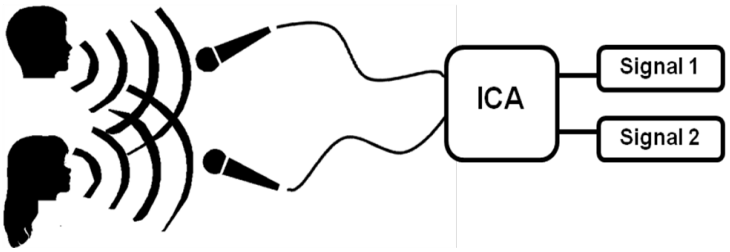


Figure 3.2

In order to implement our design to enable the separation of two audio signals, we require two microphones and a processing computer with audio output capabilities. When two microphones are present in an environment with two sources, then they will record a mixing of both these signals, weighted by coefficients base on distance away from the microphone (remember that as a signal is further away from a recording source, the quieter it will be).

The mixed signal inputs of these microphones must then be imported into a processing computer that is able to run code based in C or MatLab (or potentially any other language, but the most efficient algorithm is run in MatLab and C).

The model for these mixed signals can be represented in matrix notation by:

$\mathbf{x} = \mathbf{As}$

As we can see, the signal vector is multiplied by some **mixing matrix** A. Therefore to isolate the signals **s** based on the mixed signals **x** we must find an inverse matrix A-1. To do this we implement a piece of Matlab code known as **FastICA**. The math behind the algorithm is explained elsewhere but the most important piece of code, the iteration to find the inverse mixing matrix, is displayed here:

```
% Take a random initial vector of length 1 and orthogonalize it
% with respect to the other vectors.
if initialStateMode == 0
w = randn (vectorSize, 1);
elseif initialStateMode == 1
w=whiteningMatrix*guess(:,round);
end
w = w - B * B' * w;
w = w / norm(w);

wOld = zeros(size(w));
wOld2 = zeros(size(w));

% This is the actual fixed-point iteration loop.
% for i = 1 :  maxNumIterations + 1
i = 1;
```

[2]This content is available online at <http://cnx.org/content/m15639/1.2/>.

```
   gabba = 1;
   while i <= maxNumIterations + gabba
   if (usedDisplay > 0)
   drawnow;
   end

   % Project the vector into the space orthogonal to the space
   % spanned by the earlier found basis vectors.  Note that we can do
   % the projection with matrix B, since the zero entries do not
   % contribute to the projection.
   w = w - B * B' * w;
   w = w / norm(w);

   if notFine
   if i == maxNumIterations + 1
    if b_verbose
    fprintf('\nComponent number %d did not converge in %d iterations.\n', round,
maxNumIterations);
    end
    round = round - 1;
    numFailures = numFailures + 1;
    if numFailures > failureLimit
    if b_verbose
    fprintf('Too many failures to converge (%d).  Giving up.\n', numFailures);
    end
    if round == 0
    A=[];
    W=[];
    end
    return;
    end
    % numFailures > failurelimit
    break;
```

This iteration will guess a row of the **demixing** matrix (w in the code) and then run through a loop until it finds a projection that agrees with the statistical analysis behind the decoding.

After running on the supplied mixed signals, the program will output what it thinks are the two original sources and the **demixing** matrix. We can then use these outputs to complete a variety of tasks such as removing noise from the original signals, matching the original signals with other signals to detect base elements of the mixed signal, or even just simply outputting the original signals through a speaker system.

3.3 Blind Source Separation Via ICA: Math Behind Method[3]

3.3.1 The Math of ICA

The Independent Components Analysis algorithm allows two source signals to be separated from two mixed signals using statistical principles of independence and nongaussianity.

[3] This content is available online at <http://cnx.org/content/m15708/1.1/>.

3.3.1.1 Defining the Problem

ICA assumes that the value of each source at any given time is a random variable. It also assumes that each source is statistically independent, meaning that the values of one source cannot be correlated to values in any of the other sources.

With these assumptions, ICA allows us to separate source signals from mixtures of these source signals. The algorithm requires that there be as many sensors as input signals. For example, with three independent sources and three mixtures being recorded, the problem could be modeled as:

$$\begin{aligned} x_1(t) &= as_1(t) + bs_2(t) \\ x_2(t) &= cs_1(t) + ds_2(t) \end{aligned} \tag{3.1}$$

Using matrix notation, the problem can be generalized to any number of mixtures. For some number of sources n to be identified, n mixtures would need to be recorded.

$$x = As \tag{3.2}$$

The goal of blind source separation using ICA is to invert this procedure; that is, given the mixtures $\mathbf{x}$ as inputs, ICA finds $\mathbf{s}$. Because the mixing matrix $\mathbf{A}$ is square, we can write the reverse procedure as

$$x = A^{-1}s \tag{3.3}$$

or, if we define $\mathbf{W}$ to be equal to the inverse of $\mathbf{A}$,

$$x = Ws \tag{3.4}$$

is an equivalent expression of the problem at hand.

3.3.1.2 Isolating the Independent Sources

The Central Limit Theorem provides the key to unlocking the mystery matrix $\mathbf{W}$. The central limit theorem says that a sum of independent random variables can be approximated by a normal curve. The greater the number of variables summed, the more normal, or gaussian, the distribution. Since each of the mixtures being received by the sensors represents a linear combination of samples from each source in $\mathbf{s}$, the distribution of the mixed signals is more gaussian than either of the two independent sources (by the central limit theorem).

In order for the ICA algorithm to use this principle, the algorithm needs a way of determining how gaussian a particular signal is. There are two main quantitative measures of nongaussianity. The first of these measures is kurtosis, which measures the "spikiness" of a signal. Kurtosis is 0 for gaussian random variables, positive for random variables that are more spiky than gaussian variables, and negative for random variables that are flatter than gaussian variables. The second measure is negentropy, which measures the "simplicity" of a signal. Negentropy is also 0 for gaussian random variables.

The FastICA package that we used for our project uses both of these measures of nongaussianity to identify independent source signals. It begins by guessing a row of the matrix $\mathbf{W}$, which we can call $\mathbf{w}$. This row represents the weighting coefficients for finding one of the original source signals. It then measures the nongaussianity of the proposed independent source defined by its guess of $\mathbf{w}$, and finds the gradient of nongaussianity in an n-dimensional space to determine how the coefficients in $\mathbf{w}$ should change. It then uses a projection of the gradient to create a new guess of the coefficients in $\mathbf{w}$, and continues in a cycle until the coefficients converge on certain values. Once this occurs, the resulting independent source is as nongaussian as it can be. This in turn means that it is the furthest the algorithm could get from the source being a sum, which means that one of the independent sources has been isolated.

The algorithm repeats this process for finding all the rest of the independent sources, taking care not to find the same source twice.

3.3.1.3 Ambiguities and Limitations

Just by examining the statement of the problem at the beginning of this module, two significant ambiguities arise in the ICA algorithm.

3.3.1.3.1 Indeterminate Energy

Because a scalar multiplier could be pulled out of **s** and multiplied to **A** with no change in the above equations, the ICA algorithm cannot determine the energy contained in any of the independent sources it finds. The amplitudes it gives the output components are arbitrary, and the true source signal could be one the isolated sources multiplied by any scalar multiple. This includes a negative multiple, which means that often, the output signals are also inversions of the original signals.

The seriousness of this ambiguity depends on the application. For sound signals, inversions are irrelevant because the only important part of the signal is the different between voltages, not the polarity. Gain can also be added on to sound systems to deal with the amplitude ambiguity. In other applications, such as image processing, the inability to distinguish energy is much more significant.

3.3.1.3.2 Order Ambiguity

Because the algorithm chooses coefficients of **w** at random when it searches for the sources, the isolated sources that the algorithm finds can come out in any order. So, it would take some additional processing to determine which independent sources is the one of interest to you.

3.3.1.3.3 Under-Determination

There *must* be as many sensors as there are sources in order to properly isolate the sources. If there are not enough sensors, the resulting signals will not match any of the sources, but rather will still be mixtures of multiple sources.

3.3.1.3.4 Under-Determination

ICA can only handle linear mixtures that can be represented in the form $\mathbf{x} = \mathbf{As}$. The algorithm cannot accurately guess the independent sources if the sources are out of phase in the mixtures or if the mixtures have other nonlinear features.

3.4 Blind Source Separation Via ICA: Methods and Trials[4]

3.4.1 Blind Source Separation Via ICA: Methods and Trials

3.4.1.1 Methods and Trials

During our implementation stage we attempted several different methods for blind source separation. We came up with a plan for our initial goal: to have a working real-time system; our fallback was to have preprocessed recordings and play them as .wav files during our presentation. As is the case with many projects, we ran into some problems that we were not expecting. We were forced to identify the problems and adapt to seek new manners in which to perform blind source separation. We ran many trials with a total of four different methods.

[4]This content is available online at <http://cnx.org/content/m15638/1.3/>.

3.4.1.1.1 First Trial

We began our first trial with the intent to gain a better understanding of the **Matlab** code as well as get a feel for the recording process. Shortly after beginning, we ran into our first problem: **Windows** operating system only recognizes one microphone input. Before buying software to overcome this, we looked for solutions using two laptops.

Our system included an operator, who started recording on both laptops at the same time, two microphones connected to the mono input of each laptop, as well as two speakers talking in different places in the room. We used **Audacity** to record and convert the microphone inputs to .wav files. **Matlab** was used to trim the two recordings to the same size and run them through FASTICA.m.

The two decoded signals were very noisy. There were also remnants of the other speaker in each of the two decoded signals. After trying different rooms, different levels of noise, different types of microphones (mono and stereo), different speakers, and different locations of speakers and microphones, we could not eliminate these imperfections. We were not happy with these results so we decided to try a different method.

3.4.1.1.2 Second Trial

In our second trial, we aimed to eliminate the noise and remnants of the other speaker in our decoded signals. We attributed these problems to our signals being asynchronous. In order to solve this problem, we sought a method to record simultaneously on one computer. Using **Garageband** on a Mac, we recorded two microphone inputs onto the two channels of a stereo recording. We converted the file to a .wav file and separated the two channels in Matlab.

After running the two signals through the **Matlab** code, we encountered the same problems (noise and other speaker interference). We tested while changing all the variables as in our first trial and received similar results. This could have been from **Garageband**'s method of recording to separate channels on stereo.

3.4.1.1.3 Third Trial

For our third trial, we implemented nearly the same system as the second trial. We were mainly testing to observe if **Labview** would have better results using the method of saving to separate .wav files instead of separate stereo channels.

We wrote the following **Labview** module:

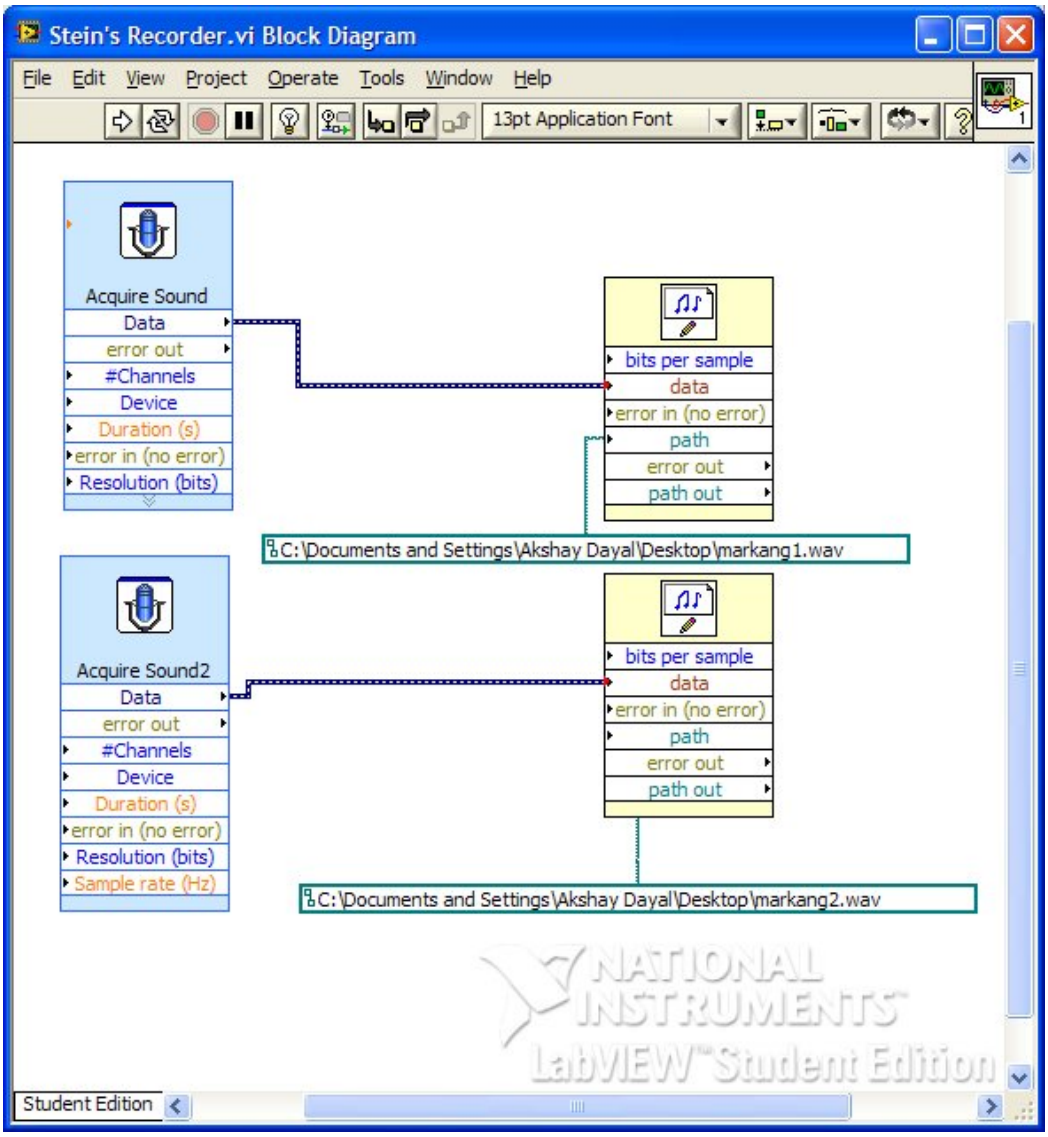


Figure 3.3

Our results were the same as the first two trials. With this method, it seemed as though it would not be possible to have the recordings asynchronous, but we were still receiving the same problems. This could be either due to limitations in our recording equipment, or due to nonlinear combinations of the signal.

3.4.1.1.4 Fourth Trial

In our final trial, we recorded two signals separately using **Audacity** and one microphone input in a quiet room. We manually combined these signals linearly in **Matlab** and after running the combined signals through our code, finally achieved a near perfect separation of the two signals. We concluded that our issues in the first three trials were not due to phasing (eliminated in third trial), but were in fact due to non linear

combinations of the signals, and also possibly to under determination caused by room noise being the third input. We saved these recordings to use in our presentation as this process is too tedious to perform in a real time demonstration.

3.5 Blind Source Separation Via ICA: Audio Demonstration[5]

3.5.1 Audio Demonstration

The following signals are examples of audio files that we ran through our code. The two signals are linearly mixed, then run through **Matlab** code to recover the original signals.

3.5.1.1 Demonstration 1

This is an example of ICA being used to separate out individual voices from two people simultaneously talking. The first two are the linearly combined signals; the last two are the results of the ICA code.

This is an unsupported media type. To view, please see http://cnx.org/content/m15712/latest/Mixed Signal 1A.wav

This is an unsupported media type. To view, please see http://cnx.org/content/m15712/latest/Mixed Signal 1B.wav

This is an unsupported media type. To view, please see http://cnx.org/content/m15712/latest/Recovered Signal 1A.wav

This is an unsupported media type. To view, please see http://cnx.org/content/m15712/latest/Recovered Signal 1B.wav

3.5.1.2 Demonstration 2

This is an example of ICA being used to separate out individual songs that are playing at the same time. The first two are the linearly combined signals; the last two are the results of the ICA code.

This is an unsupported media type. To view, please see http://cnx.org/content/m15712/latest/Mixed Signal 2A.wav

[5]This content is available online at <http://cnx.org/content/m15712/1.1/>.

This is an unsupported media type. To view, please see http://cnx.org/content/m15712/latest/Mixed Signal 2B.wav

This is an unsupported media type. To view, please see http://cnx.org/content/m15712/latest/Recovered Signal 2A.wav

This is an unsupported media type. To view, please see http://cnx.org/content/m15712/latest/Recovered Signal 2B.wav

3.6 Blind Source Separation Via ICA: Signal Processing Applications[6]

3.6.1 Blind Source Separation via ICA:

3.6.1.1 Applications: Audio/Visual Signal Processing

One of the most practical uses for blind source separation (**BSS**) is in the audio world. It has been used for noise removal without the need of filters or Fourier transforms, which leads to simpler processing methods. There are various problems associated with noise removal in this way, but these can most likely be attributed to the relative infancy of the BSS field and such limitations will be reduced as research increases in this field.

It isn't hard to imagine many situations where you receive a mixed audio signal and would love to be able to isolate the various components. Now such situations can be resolved through the implementation and algorithm presented in this project. Instruments can be separated from each other in a concert, background noise can be eliminated in any environment, and two voices from a conversation can be completely separated, as shown by figure 1.

[6] This content is available online at <http://cnx.org/content/m15640/1.3/>.

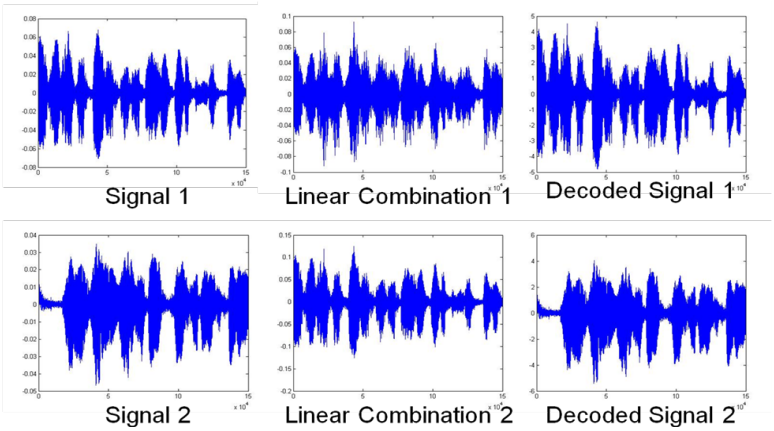


Figure 3.4

Figure 1

Voices could potentially be encrypted in this way, mixing together signals so that the original sources are indistinguishable, but all you have to do to decrypt is break the signal down into the independent components. Jamming signals sent to cell phone towers could be isolated and completely removed, something that Special Operations forces around the world are apparently using to make sure their communications can't be impeded.

In fact such transmission schemes such as code division multiple accesses (**CDMA**) can be implemented with BSS instead of the traditional orthogonality, another function of BSS that directly applies to the industry around cellular phones.

So long as you meet the required assumptions to be able to process data (that the components you are trying to isolate are indeed independent and that you have as many mixed signals as you have sources) you can process almost anything. The following images show BSS applied to **visual** examples of pictures. The format doesn't matter; ICA can be run on anything, and will always provide correct results as long as the sources are independent to begin with.

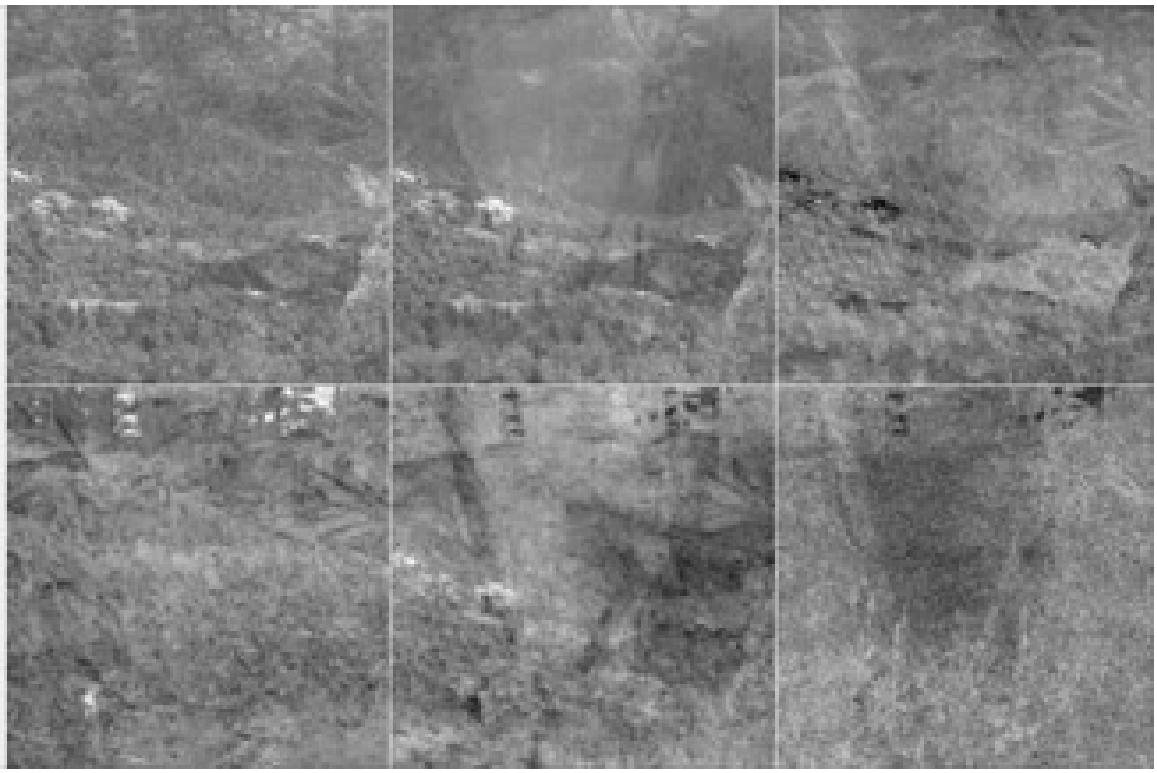


Figure 3.5

Mixed image signals

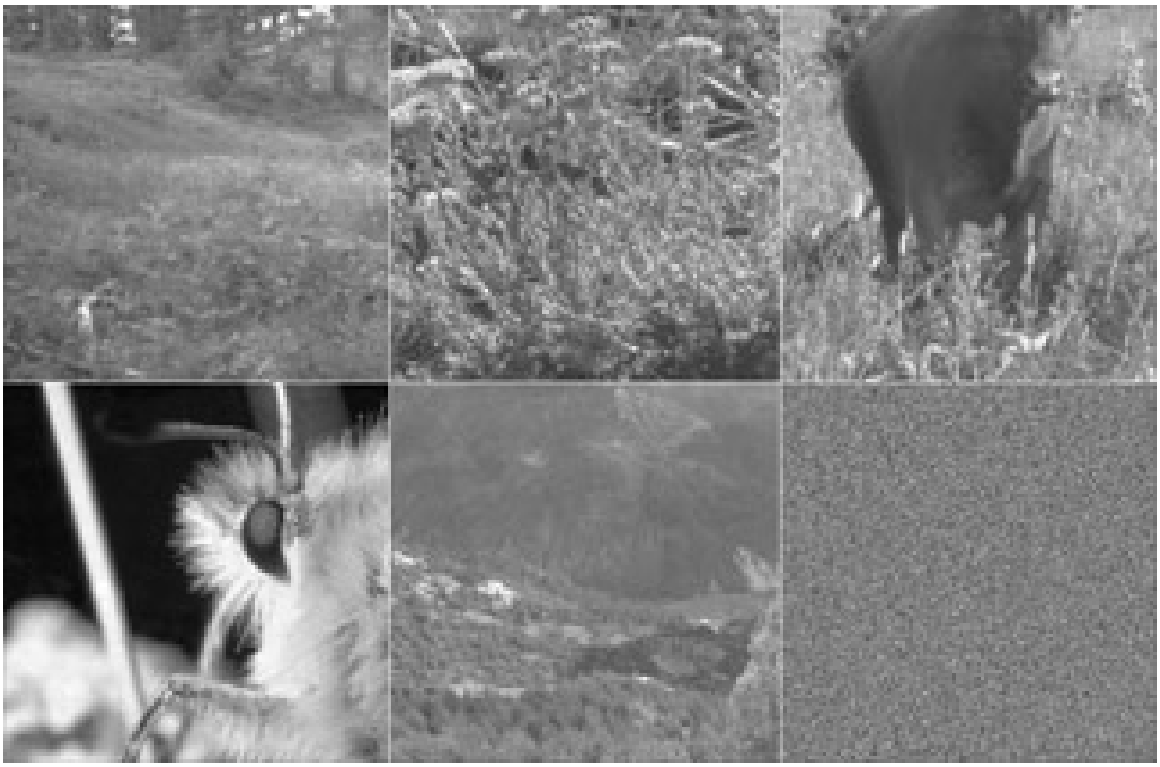


Figure 3.6

Separated Source Signals

3.7 Blind Source Separation Via ICA: Medical Imaging Applications[7]

3.7.1 Blind Source Separation via ICA

3.7.1.1 Applications: Medical Signal Processing

One of the most innovative fields in which blind source separation applies to is in medical signal processing. Medical imaging will benefit immensely from new techniques of analysis based on ICA to reduce noise, enhance images, and estimation of neuronal brain source signals.

One of the greatest challenges in neurophysiology is to analyze the physiological changes in different parts of the brain in a non-invasive method. Currently, electroencephalography (EEG) and functional magnetic resonance imaging (fMRI) are two methods that analyze neural activity. In EEG analysis of neural activity, images of the brain are taken by measuring the electric potential caused by neuron firing. In each region of the brain, non-local signals, generated in other areas, combine to form a single potential.

However, the sought after brain source signals are typically weak, non-stationary signals that are often distorted by interference, large amounts of noise, and continuous brain activity. Such noise interference consists of eye or muscle movements. The electrodes and sensors used for EEG picks up the superimposed noise signals, which corrupt the brain source signals. Current methods do not have the capabilities to

[7] This content is available online at <http://cnx.org/content/m15701/1.1/>.

separate some brain source signals based on the observed image because they are such weak signals that are often filtered out by the EEG recording systems.

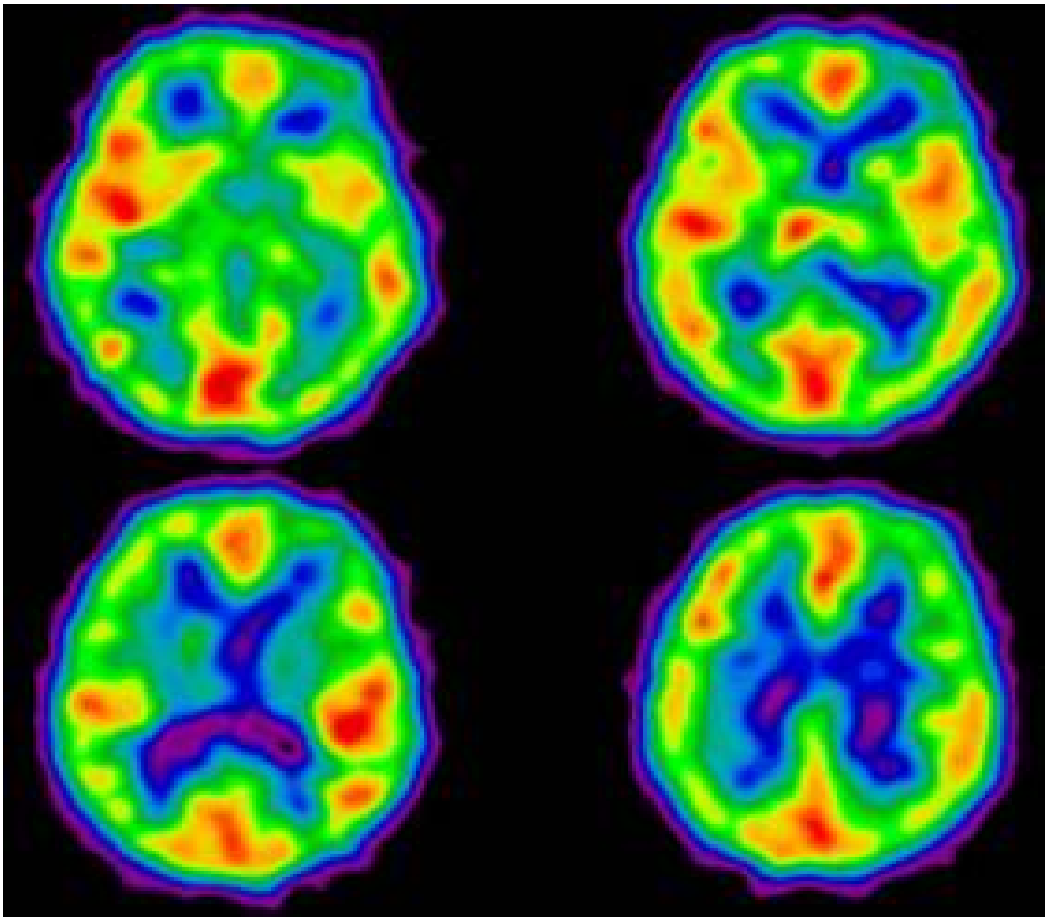


Figure 3.7

Using ICA, medical technicians will be able to separate out noise and interfering artifacts from the raw data. They will be able to recover neuronal brain source signals and trace them back to their origins. In addition, the signal-to noise ratios (SNR) will be improved as well as the spatial resolution. Medical technicians will be able to compare healthy brain signals to ones generated by an unhealthy brain.

One application example is to the early detection of Alzheimer's disease. EEG incorporating ICA will be able to detect the slight changes caused by the prevalence of Alzheimer: dementia. The brain signals of dementia are very small and difficult to detect because they are located deep within the brain. However, using blind source separation, doctors will be able to separate out the small independent signals and doctors will be able to analyze any abnormalities. Since there is no current treatment for Alzheimer's disease, early detection of dementia will be very useful in diagnosis as well as treatment.

3.8 Blind Source Separation Via ICA: Conclusion[8]

3.8.1 Conclusions and Opportunities for Further Study

3.8.1.1 Conclusions

Our implementation of blind source separation using the FastICA algorithm led us to overall successful results. We were reliably able to record two independent sound signals, digitally mix them to create mixtures, and then recover the original signals using ICA. Hardware deficiencies and phase effects between inputs disallowed us to do more real-time processing (where two microphone signals would simultaneously be recorded); however, this also helped us understand the shortcomings of the ICA algorithm. As effective as the ICA algorithm was under the right conditions, it could not work with nonlinear mixtures.

3.8.1.2 Opportunities for Further Study

Blind source separation, and ICA in particular, are currently cutting-edge fields of research and development in the signal processing world, so opportunities for further study abound. Every member of our group was fascinated by the real-world applications of ICA, especially those applications related to medical imaging. Within medical imaging, a great deal of research needs to be done on the most effective way to capture signals from the body so that ICA can return signals useful to the medical world. Outside of the medical imaging realm, applications of ICA like CDMA communication and image processing are still in their infancy, and a great deal of research could be done to understand and improve these applications. And of course, finding new applications of the ICA algorithm would be very interesting research.

Another opportunity for further study would be in finding ways to identify each of the resulting independent components. For example, in applications where ICA is used to remove interference and background noise, an important problem is that of determining which component is the desired original signal and which component represents noise.

3.9 Blind Source Separation Via ICA: The Team and Thanks[9]

3.9.1 Blind Source Separation via ICA

3.9.1.1 The Team and Thanks

3.9.1.2 The Team

- Akshay Dayal, ECE Department, Akshay.Dayal@rice.edu[10]
- Mark Eastaway, ECE Department, Mark.Eastaway@rice.edu[11]
- Angela Qian, ECE Department, Angela.Qian@rice.edu[12]
- John Steinbauer, ECE Department, John.J.Steinbauer@rice.edu[13]

3.9.1.2.1 Thanks and Recognition

The team would also like to thank Professor Richard Barniuk and Mark Davenport, who helped jumpstart us on our project. We also would like to thank Michael Dye and the Rice University ERC for providing the microphone hardware we used to implement our project.

References

[8] This content is available online at <http://cnx.org/content/m15706/1.1/>.
[9] This content is available online at <http://cnx.org/content/m15703/1.1/>.
[10] Akshay.Dayal@rice.edu
[11] Mark.Eastaway@rice.edu
[12] Angela.Qian@rice.edu
[13] John.J.Steinbauer@rice.edu

- Gävert, H., Hurri, J., Särelä, J., and Hyvärinen, A. (2005). The FastICA Package for MATLAB[14] . Free Software Foundation.
- Hyvarinen, A. and Erkki, O.(2000). Independent Component Analysis: Algorithms and Applications[15] . Neural Networks, 13(4-5): 411-430.
- Cichocki, A. (2004). Blind Signal Processing Methods for AnalyzingMultichannel Brain Signals. International Journal of Bioelectromagnetism2004, Vol. 6, No. 1[16] .
- Jourjine, A., Rickard, S., Yilmaz, O. (2000). Blind Separation of Disjoint Orthogonal Signals: Demixing N Sources from 2 Mixtures. IEEE.
- Fiori, S. (1999). Entrophy Optimization by the PFANN Network: Application to Blind Source Separation. Network: Cornput. Neural Syst. 10 171-186.
- Meyer-Base, A., Gruber, P., Theis, F., and Foo, S. (2005). Blind Source Separation Based on Self-Organizing Neural Network. Science Direct.

[14] http://www.cis.hut.fi/projects/ica/fastica/
[15] http://cnx.org/content/m15703/latest/file:///var/local/word-harvest/good/4.%09http:/www.cvmt.aau.dk/education/teaching/f05/INF6/Fea
[16] http://ijbem.k.hosei.ac.jp/volume6/number1/index.htm

Chapter 4

Remote Sound Detection Using a Laser

4.1 Introduction[1]

Our project provides a setup for detecting sound remotely by reflecting a laser beam off a hard surface, usually a window. Any sound that is near a window causes the window to move, and this technology takes advantage of that. It allows sounds to be heard from very far away because the sound information travels using the light as a medium, instead of the pressure waves of sound, it attenuates much less quickly. Another interesting thing to note is that the sound information is traveling at the speed of light instead of the speed of sound, so the information arrives more quickly than it would in a normal situation.

As one might imagine, this has interesting surveillance applications. This technology is currently being used by the CIA and many other surveillance-related organizations to eavesdrop. However, the main difference is that they use phase detection and infrared lasers while we use amplitude detection and a ruby laser for cost purposes. The phase modulation is much more accurate and much less noise prone, but requires a more complicated setup and was also not the goal of our project. The infrared laser is also useful in real-world scenarios because of the fact that it is invisible; the ruby laser might cause the surveillance subject to realize that they are being watched.

4.2 Setup[2]

The setup itself is rather simple. It just consists of a laser pointer that is pointed at a window or any reflective hard surface. The sound vibrations cause the hard surface to act as a diaphragm and vibrate along with the sound. There is also a photodetector that picks up the light and measures the intensity, which is sent to a computer. We used the audio-in line on a laptop to input the changes in voltage measured by the photodetector to the computer. Then, on the computer we performed all our signal processing through Matlab 7.0 and Labview.

This vibration in the reflective diaphragm causes the laser beam to change direction slightly, which causes the intensity that is perceived by the photodetector to change. Our first laser pointer was more focused and would cause our photodetector to maximize its output (causing railing or clipping) which would make changes undetectable. To rectify this situation we moved the laser beam slightly off the photodetector so that it was only partially hitting. Causing it to rail then moving it slightly off the photodetector resulted in the best sounding signal. The resulting changes in intensity are then sent through the audio line.

There are several problems that must be dealt with in the implementation of this laser microphone that are listed as follows:

[1] This content is available online at <http://cnx.org/content/m15680/1.1/>.
[2] This content is available online at <http://cnx.org/content/m15682/1.1/>.

1. In addition to the laser light, ambient light is picked up by the photodetector. This ambient light may change and vary randomly, or may be synched with the 60Hz frequency of the electrical grid if the lights are florescent, which most of them are. Most of the ambient light was removed by simply adding a dark long tube for the laser to pass through before it reached the photodiode at the end. Implementing this blocks out a large percentage of the light, as it is not aligned directly with the tube and therefore cannot reach the photodiode. The back side of the tube must also be protected from light, so an opaque cloth covering was used to allow the wires attached to the photodiode to have freedom of movement.
2. There exists basic electrical noise on the circuit. This noise comes from both fluorescent lights, and from EMF noise produced by the power grid being picked up on the wires. The noise is at 60Hz, 120Hz, and other harmonics of 60Hz.
3. most importantly, there are significant changes in the sound signal due to the properties of the window. The window has properties such as the size, thickness, and the choice of material. These properties alter how the window vibrates when it receives the sound signal from the air. The window can be treated as a filter to the sound, as it resonates with certain frequencies and dampens others. We solved this problem with a complex set of inverse filters that will be explained in detail later in this document.

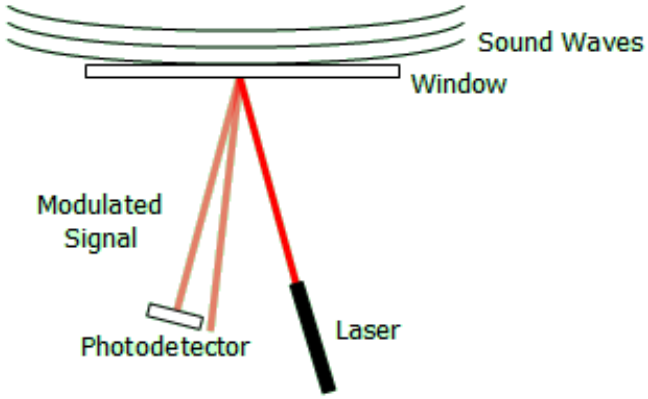


Figure 4.1

4.3 Implementation[3]

The hardware implementation of the Laser Microphone is relatively simple and can be done at a minimal cost.

[3]This content is available online at <http://cnx.org/content/m15679/1.1/>.

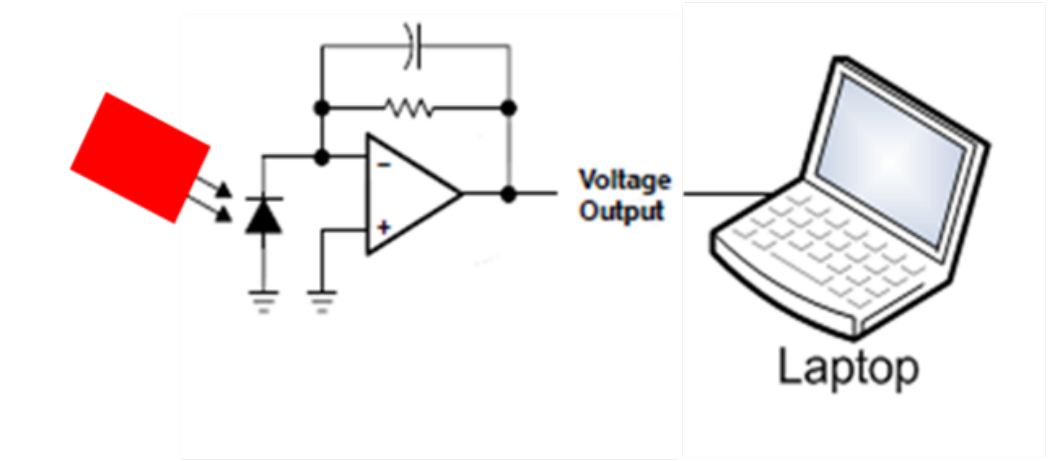


Figure 4.2

4.3.1 Laser/Photodetector:

The laser we used was a simple presentation laser pointer that outputted a red beam at approximately a 650nm wavelength. To receive the signal, we used a low cost photodetector (TSL12s), which is simply a photodiode and a trans-impedance amplifier combined together in a single package. The peak of the photodetector's spectral response characteristics coincide with the output wavelength of the laser pointer.

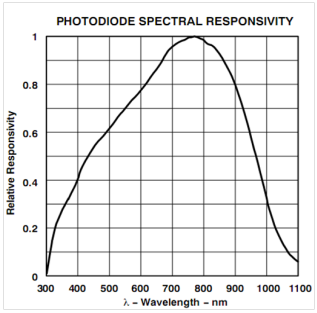


Figure 4.3

4.3.2 Detection Unit:

The laser capture setup was a cardboard tube with a small hole in one end for a photodetector in order to obstruct as much ambient light as possible. A power supply was used to create the 5 volt supply voltage for the photodetector and a spliced 1/8" phono jack connecter was connected to the outputted signal.

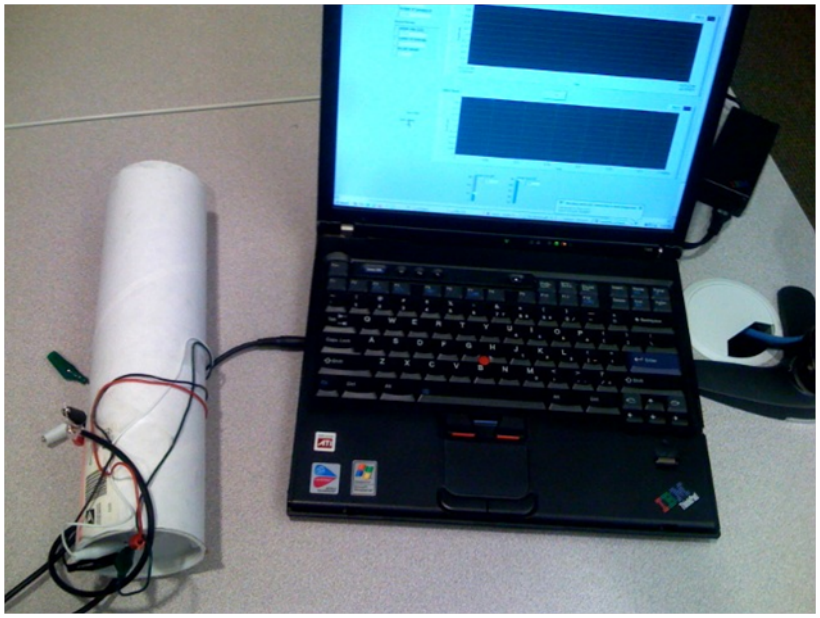


Figure 4.4

4.3.3 DAC:

The Digital to Analog converter that we use to digitize the signal for further software processing is the mic-in jack on a laptop. Using this 22.05 kHz DAC, we are able to cheaply and properly sample the 3.6 kHz speech signal while following the Nyquist criterion and thus avoid any aliasing effects.

4.4 Inverse Filter[4]

We observed that the system did not transmit sound information perfectly, and transmitted speech signals suffered some distortion. This distortion happens for two reasons: (1) the physical properties of the glass cause it to respond differently to different frequencies, and (2) low-frequency vibrations caused by air-conditioning systems and other building vibrations are constantly present in the window. We attempted to compensate for this observed distortion by building an inverse filter. We accomplished this in three steps:

4.4.1 Step 1: Measure the Frequency Response

In order to accurately model the system, we needed to measure its frequency response. We blasted a 30-second sound clip of pure white noise at the window and recorded the signal measured by the detection unit. Since we knew the input of the system (the white noise) had a completely flat spectrum, the output's spectrum should represent the frequency response. To compute the spectrum of the output (the recorded signal), we windowed portions of the signal using a Hamming window, computed the FFT's of each windowed portion, and then averaged the FFT's. This average FFT represents the frequency response of our system.

[4]This content is available online at <http://cnx.org/content/m15685/1.1/>.

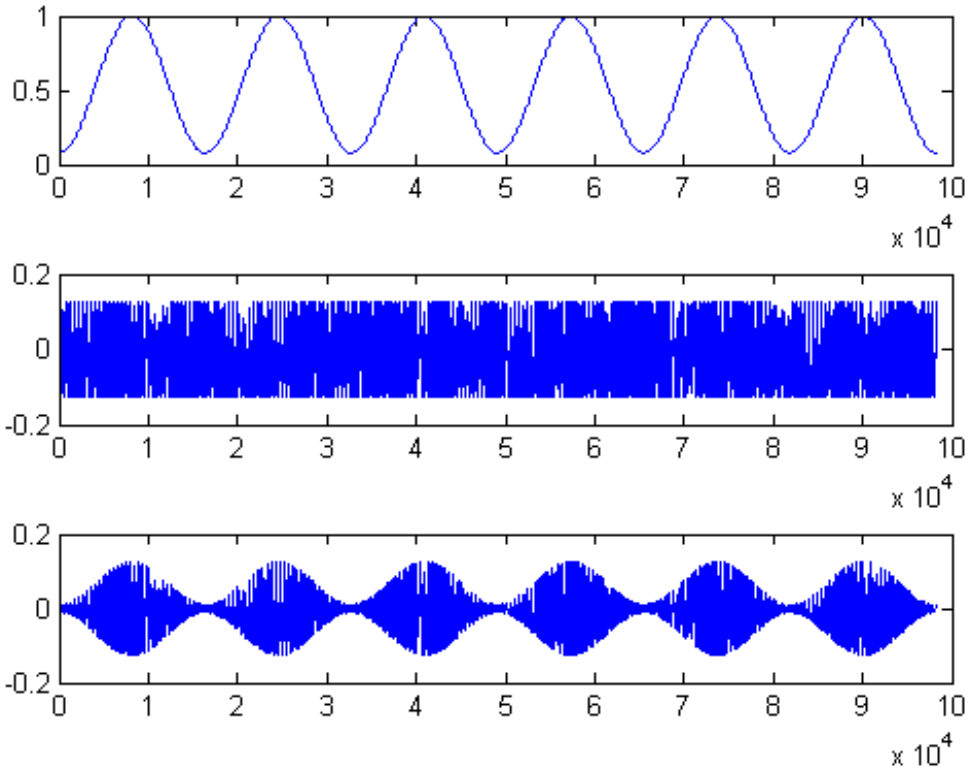

1
0.5
0
0 1 2 3 4 5 6 7 8 9 10
x 10^4
0.2
0
-0.2
0 1 2 3 4 5 6 7 8 9 10
x 10^4
0.2
0
-0.2
0 1 2 3 4 5 6 7 8 9 10
x 10^4

Figure 4.5

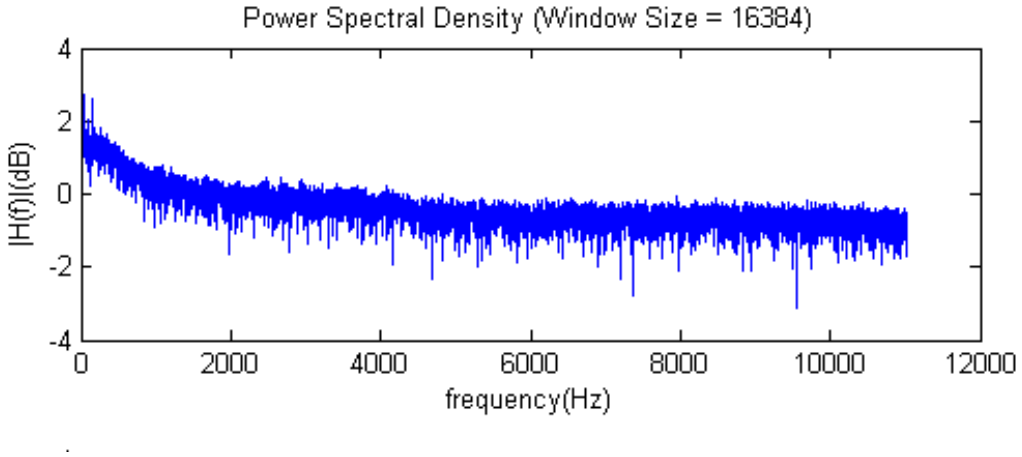


Figure 4.6

The plot shows some strong low-frequency vibrations in the window. We attributed these to the air-conditioning unit in the building and to other random vibrations in the environment. We also noticed that the window responded better to low frequencies than to high frequencies. This could be a result of the physical properties of the glass as well as the physical dimensions of the window.

4.4.2 Step 2: Model the System

Once we had a good idea of the system's frequency response, we attempted to model the system using a linear prediction filter. We used a linear prediction filter because it made the inverse filter simple to implement, and it guaranteed that the inverse filter would be inherently stable and have a linear phase response. A linear prediction filter estimates its next output by the current input and a linear combination of n previous outputs:

$$y[n] = g \cdot x[n] + \sum_{k=1}^{N} a_k \cdot y[n-1]$$

Figure 4.7

The first step to building this filter is to compute the autocorrelation coefficients of the recorded signal. The autocorrelation coefficients are a measure of the correlation between samples of the signal. Since the filter must accurately estimate the output based on previous outputs, it must preserve the correlation between samples. One autocorrelation coefficient r[i] can be expressed as:

$$r_i = \sum_{n=0}^{N-i+1} s[n]s[n+i]$$

Figure 4.8

The next step to building the filter was to compute the filter coefficients. We used a recursive algorithm called Burden's Algorithm to do this. We set the first coefficient a[0] = 1 and then compute the other coefficients recursively:

$$K_i = -\frac{r_i + a_1 \cdot r_{i-1} + \cdots + a_{i-1} \cdot r_i}{E_{i-1}}$$

$$E_i = (1 - K_i)^2 \cdot E_{i-1}$$

$$a_i = K_i$$

Figure 4.9

We could perform this recursion as many times as we needed to compute the desired amount of coefficients. We wrote a MATLAB program to perform the algorithm N times on the windowed signal to generate N coefficients. We used these coefficients in the feedback branches of the filter. We found that we could accurately model the system using a linear prediction filter with 50 coefficients. The frequency response of this filter has a similar shape to the measured frequency response of the system:

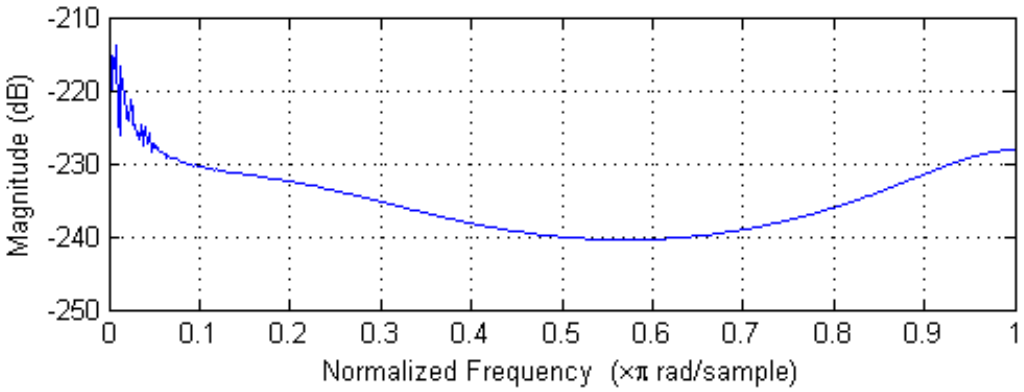


Figure 4.10

Step 3: Build the Inverse Filter

4.4.3 Step 3: Build the Inverse Filter

The linear prediction filter is simple to invert. Since it uses only the previous outputs to generate the next output, it is an all-pole filter with only feedback branches. To build the inverse filter, we used all the feedback coefficients that we generated using Burden's Algorithm as the feed-forward coefficients of the inverse filter. The frequency response of the inverse filter looks like:

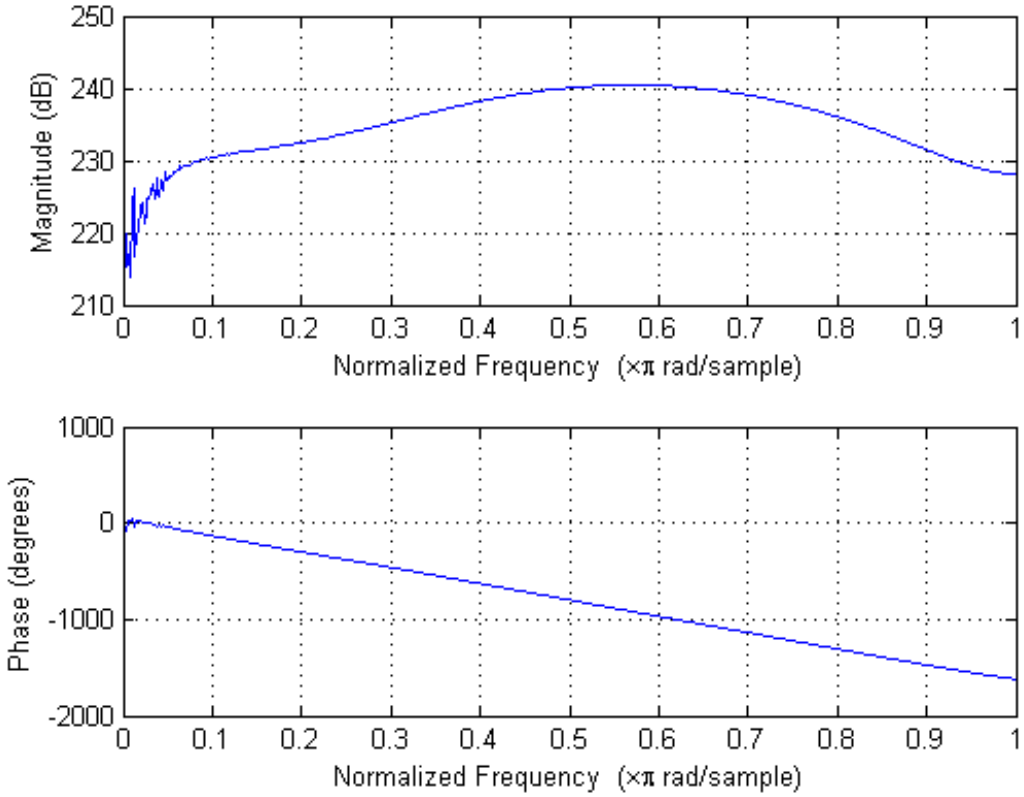


Figure 4.11

We observed that the inverse filter accurately inverted the response of the system. It successfully attenuated the low-frequency window vibrations, and it amplified the higher frequencies that the system attenuated.

4.5 Vocal Band Pass Filter[5]

After the inverse filter, we decided to isolate the speech signal to remove some of the additive noise. We accomplished this by applying a band pass filter to the recorded signal. When filtering signals, it is very useful to have an understanding of where the important information in the signal lies. With a speech signal there are a few things that we can take advantage of when attempting to filter out noise.

Speech signals generally have a distinctive envelope in the frequency domain (pictured below). After our preliminary filters, we were able to use this envelope to check and see if our output matched.

[5]This content is available online at <http://cnx.org/content/m15683/1.1/>.

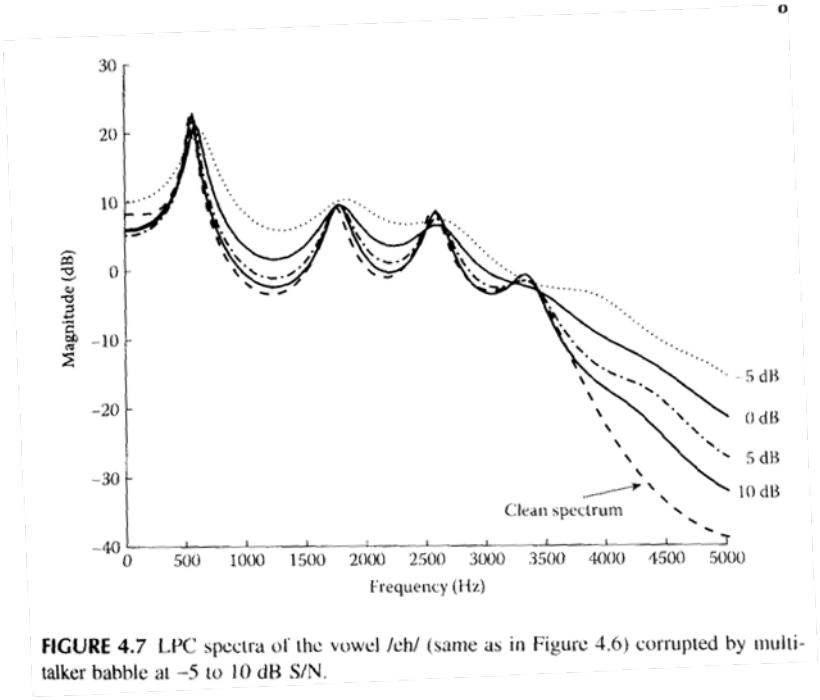


FIGURE 4.7 LPC spectra of the vowel /eh/ (same as in Figure 4.6) corrupted by multi-talker babble at −5 to 10 dB S/N.

Figure 4.12: Picture from "Speech Enhancement Theory and Practice" Philipos C. Loizou

Human speech exists within a finite frequency range. As we are trying to eliminate noise to create a more intelligible speech signal we can get rid of everything outside of this range. To do this we will use a band-pass filter. To get optimum intelligibility telephone companies will generally use a window from 300Hz-3600Hz. The military uses around 400Hz-2800Hz to get rid of more background noise. We used a band-pass filter that went from 400Hz-3600Hz. In order to efficiently design this filter to have linear phase and a finite impulse response, we utilized the Remez Exchange (or Parks McClellan) algorithm. We accomplished this in MATLAB, resulting in the frequency response shown below.

Frequency Response of Bandpass Filter

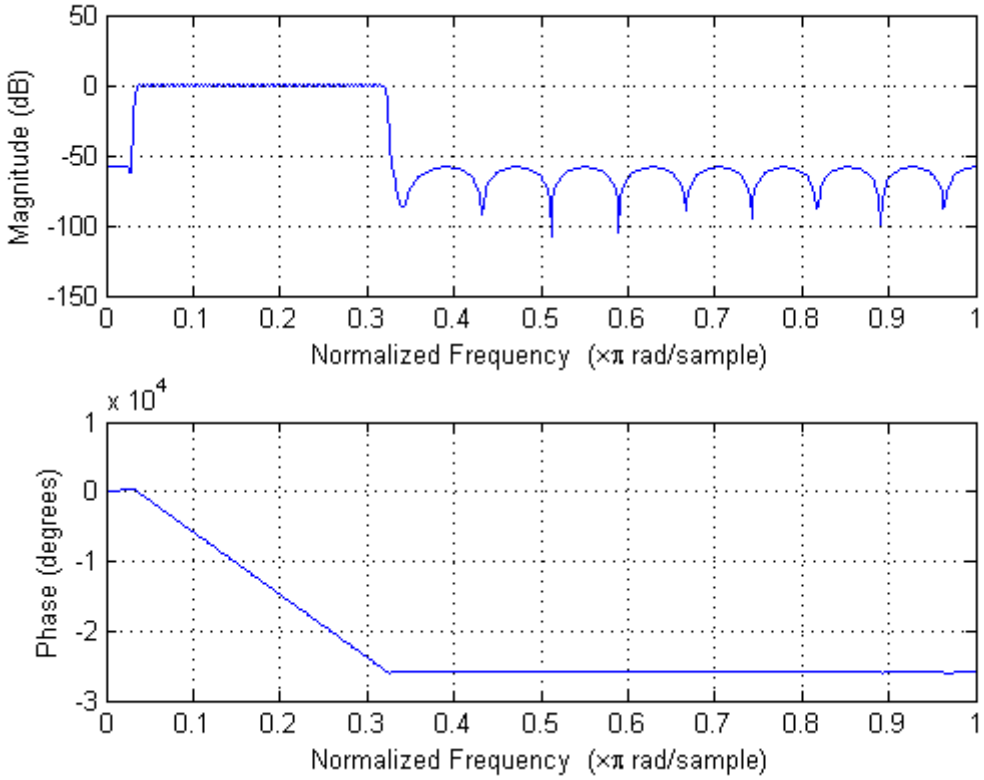


Figure 4.13

4.6 Results[6]

Both the inverse filter and the vocal band filter performed well at improving the quality of the transmitted signal by compensating for the observed distortion and removing additive noise. The inverse filter successfully boosted the high frequencies that were absorbed by the window. The vocal band filter isolated the speech portion of the signal and successfully removed much of the noise produced by the low-frequency window vibrations. The spectrum of the filtered signal appears similar in shape to the human voice spectrum in the pass band.

4.6.1 Possible Improvements

In order to improve the quality of the recorded signal, we'd like to explore ways of improving the transmission process to get better results. One method that we conceived is to modulate the laser beam at its source with a carrier frequency. We could then demodulate the recorded signal digitally. In theory, this scheme

[6]This content is available online at <http://cnx.org/content/m15684/1.1/>.

could considerably reduce the amount of additive noise in the transmitted signal by moving the transmitted speech band away from the strong low-frequency noise and into the high-frequency range.

4.6.2 Conclusion

This project was an enjoyable experience for all the members in our group. We got to experiment with a technology that was new to us, and we got to learn a lot about digital speech processing. Overall we are proud of the project.

Chapter 5

MATLAB EQ

5.1 MATLAB EQ: Problem[1]

The challenge of this project was to implement the mathematical form a graphic equalizer into an intuitive graphical user interface. The group chose to create the GUI in MATLAB. Working in MATLAB had many advantages and disadvantages. The main advantage of working in MATLAB was that MATLAB was very adept at signal manipulation. Since it was designed to manipulate matrices, MATLAB allowed for us to easily code a working bin matrix, and to plot in real time the effects of our signal alterations in time and frequency.

The main disadvantage of using MATLAB was that since MATLAB does not do things in parallel, we were not able to simultaneously take input from the soundcard to our graphic equalizer, and send an output to the soundcard.

Perhaps a better approach would have been to use LabView instead of MATLAB. LabView is designed to interface with hardware, and now it has incorporated the ability to run .m files in LabView. Therefore, in hindsight, a better way to attack the problem would have been to use LabView.

However, we did make a working graphical user interface using MATLAB that was able to create a visual output.

5.2 MATLAB EQ: Background on Equalization[2]

5.2.1 Introduction to Equalizers

This section will cover the basic principles that we used in the creation of our graphical equalizer: equalizer principles, analog to digital conversion, Nyquist-Shannon sampling Theory, and the fast Fourier transform.

5.2.1.1 Equalizers

In audio processing, equalization is the process of changing the spectrum of a sound. Equalizers employ low-pass, band-pass, and high-pass filters to modify different portions of a sound's spectrum. By changing the gains on these filters an operator is able to modify the sound. An easy example to understand would be a 'Bass Booster,' where the user adds gain to the low frequencies and thereby increases the 'bass' of a song. Generally, equalizers have ten different bands that a person is able to manipulate, but high end models can have up to thirty-one.

[1] This content is available online at <http://cnx.org/content/m15652/1.1/>.
[2] This content is available online at <http://cnx.org/content/m15655/1.1/>.

5.2.1.2 Analog to Digital Conversion

Sampling is the process of converting a continuous real time signal into a discrete time signal. The basics of sampling form the basis for much of today's music recording industry. Sophisticated digital to analog converters (DAC) can be found in music studios across the world.

A DAC works by converting voltages into a digital number representing the corresponding voltage. The digital representations of these signals can then be manipulated by a computer, introducing various effects and gains. After processing, the signal can be converted back into an analog signal using a digital to analog converter.

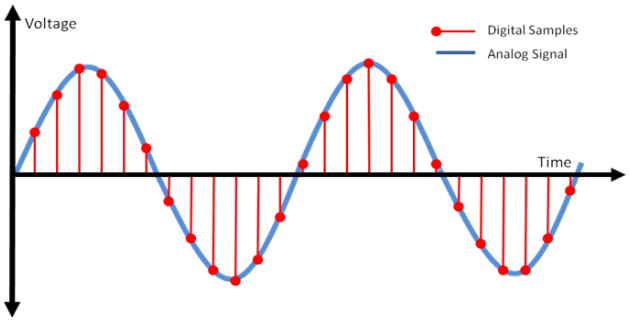


Figure 5.1: A visual example of digital sampling of an analog signal.

5.2.1.3 Nyquist-Shannon Sampling Theory and Aliasing

The Nyquist-Shannon theory states that exact reconstruction of a continuous time signal from its samples is possible if the signal is bandlimited and the sampling frequency is at least twice that bamdwodth.

For example, the human ear can hear sounds from 20-20,000Hz. This means that if we want to recreate an audio signal with the complete range of recorded frequencies we must sample the signal at a minimum of 40,000Hz. Typical music is coded and sampled at 44,100Hz which obeys the Shannon-Nyquist theorem. The reason for the extra bandwidth is due to imperfect low-pass filter effects, which is outside the scope of this module.

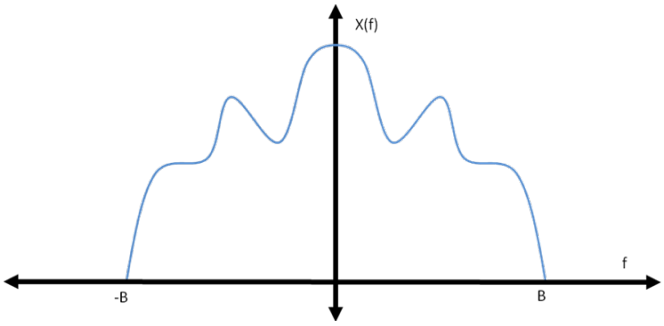


Figure 5.2: A visual example of an appropriately band limited signal.

Aliasing occurs when the Nyquist-Shannon Theorem is not obeyed, i.e. when the sampling rate is not at least two times the band limit's frequency. Aliasing is the overlap of spectrums at the band edges and results in an incorrectly reproduced signal. These overlaps corrupt the signal by adding the two spectrums from the overlapping bands.

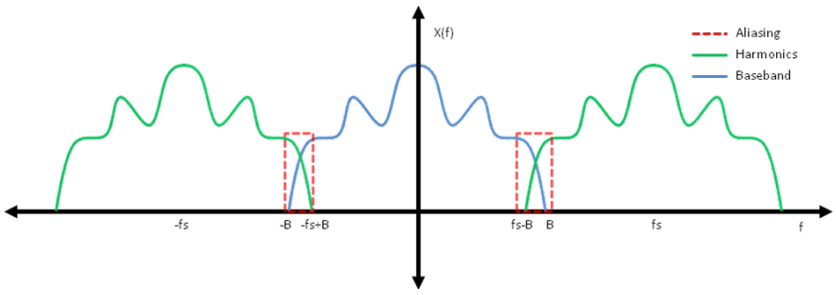


Figure 5.3: A visual example of a aliasing in a signal.

5.2.1.4 Fast Fourier Transform

To be able to process bandwidth gains, one must move the samples of the continuous time signal into the frequency regime. This is accomplished quickly and efficiently using the Fast Fourier Transform (FFT). The equation for this is as follows:

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-j2\pi}{N}nk} \qquad k = 0, \ldots, N-1$$

Figure 5.4: The Fast Fourier Transform.

This moves the signal into the frequency regime with N points. There is a problem in that you need a very long FFT to accomplish a perfect spectrum. Long FFTs are computationally expensive and are difficult to accomplish in real time, and a compromise between speed an resolution must be made.

5.2.1.5 Inverse Fast Fourier Transform

Once the signal has been manipulated in the frequency regime, you have to convert those values back into a digital representation of the signal. To do this you use the Inverse Fast Fourier Transform:

$$x_n = \sum_{n=0}^{N-1} \frac{X_k}{N} e^{\frac{j2\pi}{N}nk} \qquad k = 0, \ldots, N-1$$

Figure 5.5: The Inverse Fast Fourier Transform.

Since we have modified the spectrum, this new digital signal may have imaginary parts in it. Digital to analog converters do not interpret imaginary numbers so we must take the real part of the modified signal to be able to reproduce it.

5.3 MATLAB EQ: Sound Sampling[3]

5.3.1 Sampling Windows Audio Data in MATLAB

This module will walk you through the coding process of saving input audio data from a Windows sound card.

5.3.1.1 Setup

The first thing you need to do is set how often you would like MATLAB to access your sound card, and how fast you want it to do so. This multiplying these two values together will give you the total size of your sample space. The code below is how we initialized our data input.

```
refreshrate = .04644; % sec
samplerate = 44100; % Hz
ai = analoginput('winsound', 1); %windows addchannel(ai,[1 2]); %two channels
samplerate = setverify(ai, 'SampleRate', samplerate);
```

5.3.1.2 Trigger

Next, you must set a command structure to let MATLAB know when to access the sound data. In the case of our graphical equalizer we wanted MATLAB to sample until we told it to stop so we created an infinite trigger loop. To do this you must set the triggers on your input class.

```
ai.TimerPeriod = refreshrate;
spt = round(samplerate * refreshrate);
ai.SamplesPerTrigger = spt;
set(ai, 'TriggerRepeat', Inf);
set(ai, 'TimerFcn' , @getdata);
start(ai);
```

5.3.1.3 Storing Data and Stopping

Now that you have begun to sample the soundcard, you have to store the sampled data in a buffer for analysis. You need to flush the data in your acquisition structure or you will suffer serious memory leaks. Also, the try and catch structure allows the loading of empty values if the peekdata is empty (a type of error suppression).

```
try
```

[3] This content is available online at <http://cnx.org/content/m15657/1.1/>.

```
timesig = peekdata(ai,samples);
flushdata(ai)
catch
timesig = [];
end
```

Stopping the data acquisition is simple:

```
stop(ai)
```

5.4 MATLAB EQ: Approach: Use of Frequency Binning to Apply Frequency Gains[4]

The main focus of a graphic equalizer is to manipulate different bands of frequencies, and then reconstruct them. Therefore, to manipulate the frequencies, a group of samples were taken, and the Fast Fourier Transform of the sample set was computed. In order to maximize the efficiency of the FFT, a sample set of a power of two was used. FFTs maximize symmetry in the Discrete Fourier Transforms and are best able to do that for signals of power of two lengths. Hence a refresh rate such that 2048 samples were taken in was chosen.

Once the signal is put into the Fourier domain, we then need to separate the frequencies into the desired bands so that they can be displayed, operated on, and then be displayed in their updated state. To do this we used a binning matrix. The matrix binned according to a logarithmic Q binning pattern. This means it puts the first and 2048th frequency in the first bin, the second, third, 2047th and 2046th frequencies in the second bin and so on and so forth. The matrix below is an example of how a bin matrix would look for an eight bit Fast Fourier Transform.

0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0

Figure 5.6: Example of a bin matrix for a length 8 FFT.

The code to generate matrices of this form is below.

```
[n, bin] = histc(freqs, freqbins);
bin = bin(:,1:spt/2);

binmatrixa0 = zeros(length(n)-1, length(bin));

for k = 1:(length(n)-1)
binmatrixa0(k,:)  = bin == k ;
end

binmatrixb0 = fliplr(binmatrixa0);
binmatrix0 = [binmatrixa0, binmatrixb0]
```

[4]This content is available online at <http://cnx.org/content/m15650/1.1/>.

Once the bins are generated by multiplying the bin matrix with the FFT vector, the bins' L2 norms are generated and displayed in bar graph form. Then, all the contents of all the bins are multiplied by what the user inputs in the GUI slide bars. The L2 norms are taken again and displayed in a bar graph as well.

The multiplying the frequency bins by a constant, it is the equivalent of passing the signal through several different ideal band pass filters. This is possible, because the signals are of finite length. Therefore, a discrete sinc of this length can create a perfect discrete band pass filter.

Once these gains have been applied, the vectors representing each of the bins are recombined to reform a 2048 Fast Fourier Transform vector. The signal is then Inverse Fast Fourier Transformed to allow for the display of the time wave form, and to allow for the application of time domain effects.

The last problem that needs to be taken into account is the phase. Due to the application of non-uniform gain to the bins, there is phase added to the output waveform. This phase needs to be removed, because the equalizer is supposed to output to speakers, and speakers can only interpret real signals.

5.5 MATLAB EQ: Approach: Time-Domain and Effects[5]

5.5.1 Approach: Time-Domain and Effects

Once the Inverse Fast Fourier Transform (IFFT) has taken place, we are able to apply our time-domain effects: Reverb, Flange, and Distortion. When the effects have been implemented, we then output our modified signal into a plot where we are able to analyze the new spectrum of our signal. The last two effects, Gain and Volume, are included at the end.

5.5.1.1 Flange

Flanging is another time-domain audio effect which happens when two audio signals are mixed together, with the requirement that one of the signals is time-delayed by a very small amount of time. An analogy we like is the simultaneous playing of two tape recorders playing the same track. Then if we pause one of them for a small amount of time and then hit play, the combined audio of the two tape recorders is flanging.

In our coding, we used 15 refresh periods.

[5]This content is available online at <http://cnx.org/content/m15654/1.1/>.

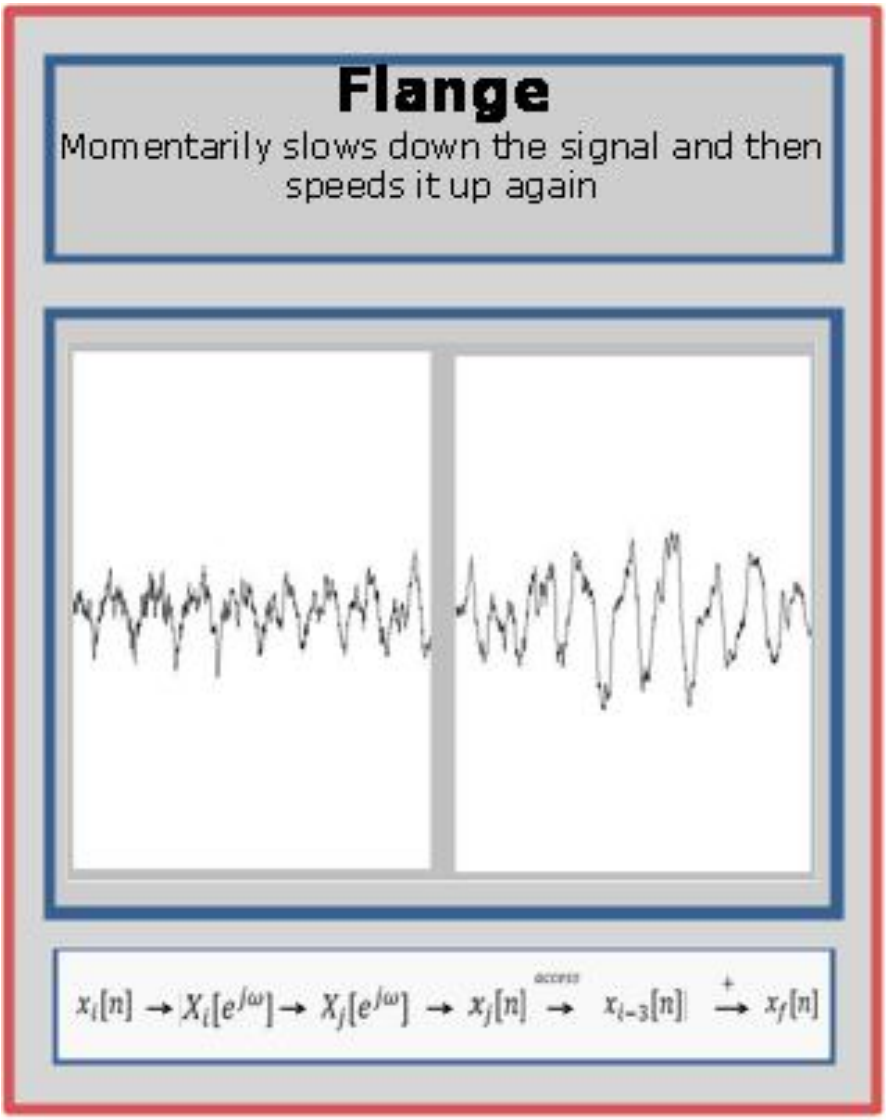


Figure 5.7

5.5.1.2 Reverb

By definition, reverberation is the lingering of sound in a given space once the original sound is removed. It is not necessarily termed "echo" as reverb is the decay of a large number of echoes when the sound is absorbed. Thus, once the sound signal stops, the reverberated reflections carry on and gradually decrease in amplitude until they zero out.

In our implementation, we used 15 consecutive refresh periods. This wasn't incredibly realistic as we could have modified our design to produce a better reverb, but this allowed our program to function smoothly without memory overflows in MATLAB.

The main visual difference between flange and reverb was the persistence of sound once the song/audio clip ended.

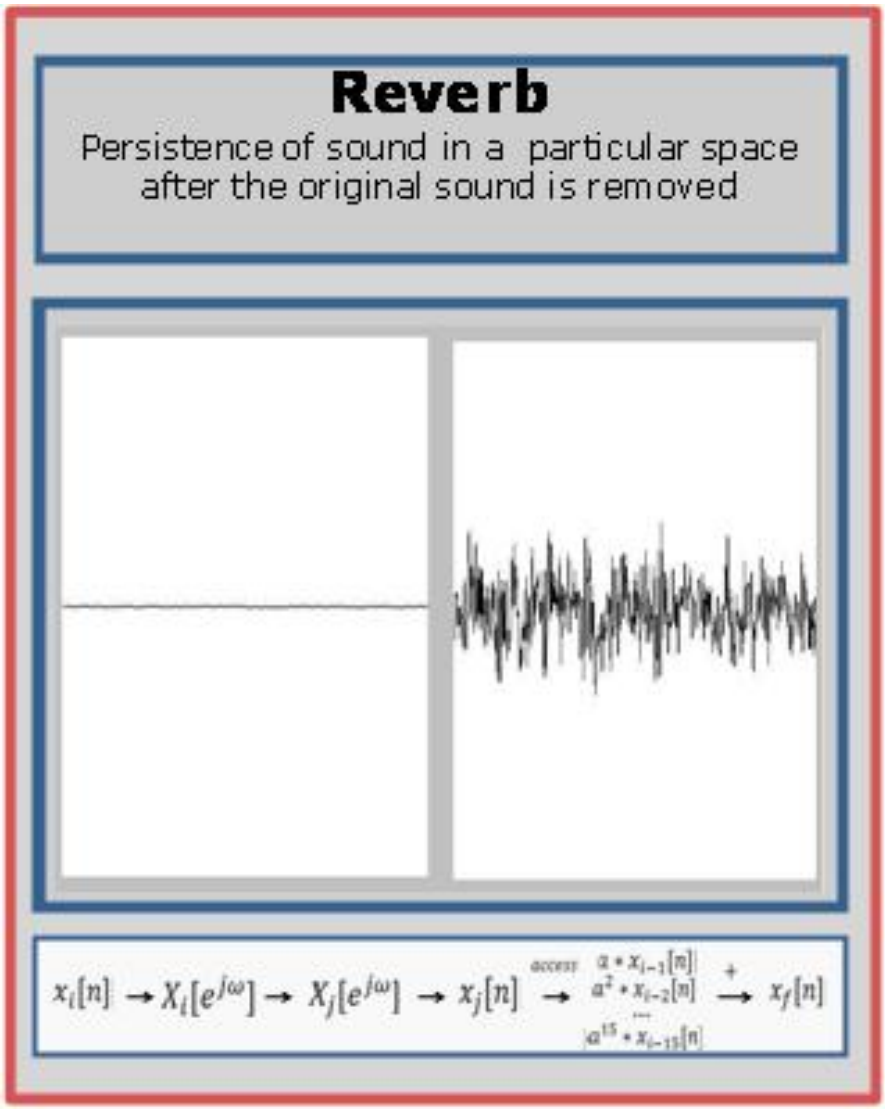


Figure 5.8

5.5.1.3 Distortion

A distortion that we introduced in our equalizer can also be referred to as "audio clipping" as we cut off signals above a certain amplitude.

This is also called hard clipping. Distortion is usually an unwanted effect except in music, such as Gabber (150-220 beats-per-minute hardcore techno) or as an electric guitar effect.

In order to guarantee a visual representation, we went ahead and clipped our signals at the 0.5 mark in amplitude.

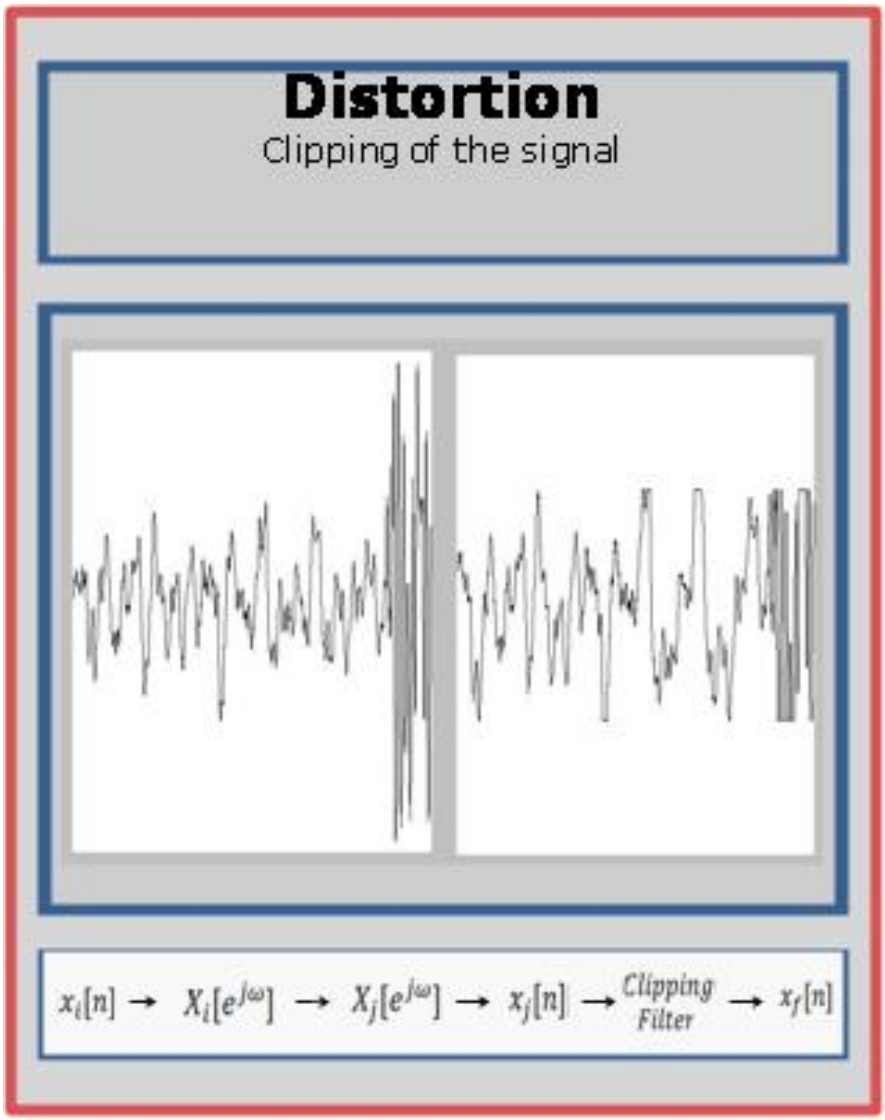


Figure 5.9

5.5.1.4 Gain

In our equalizer, we implemented a gain control which provided for the increase of our input signal's amplitude for a given frequency band. We had 10 frequency bands of which we could modify the gain levels.

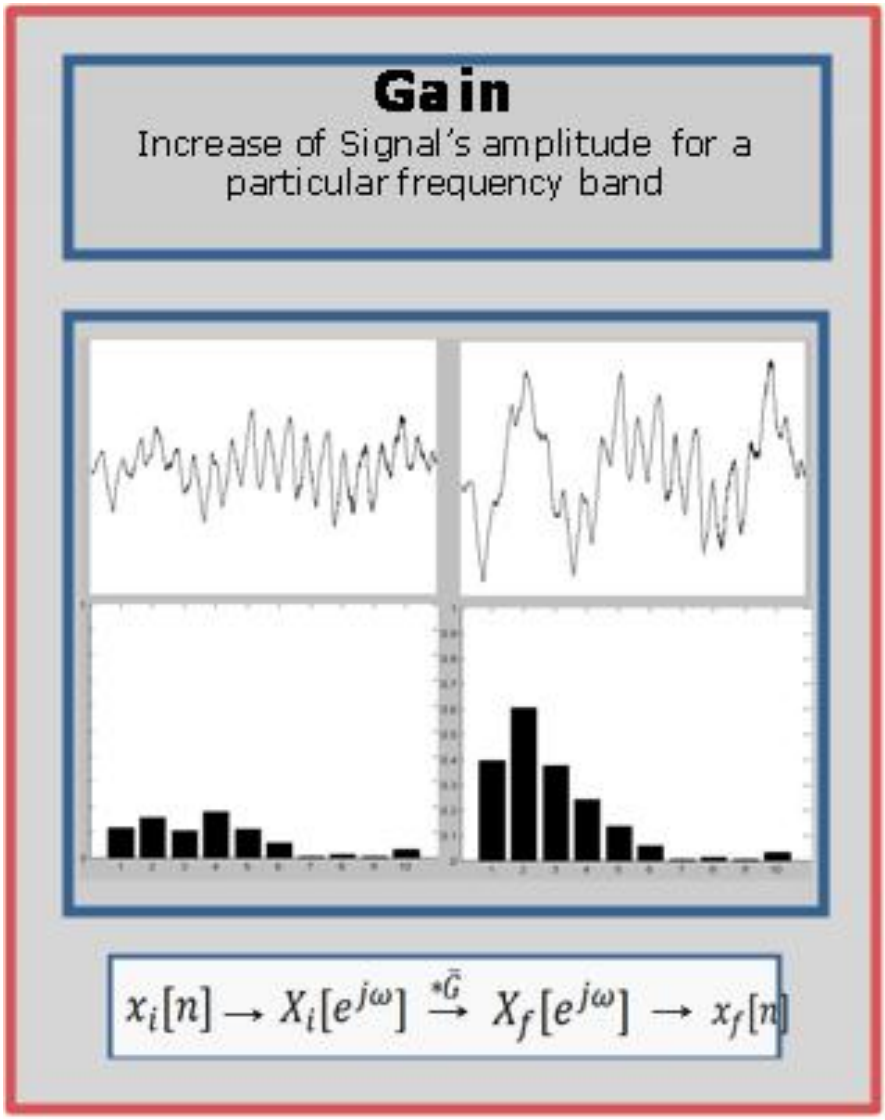


Figure 5.10

5.5.1.5 Volume

Volume is the implementation of gain for all frequencies.

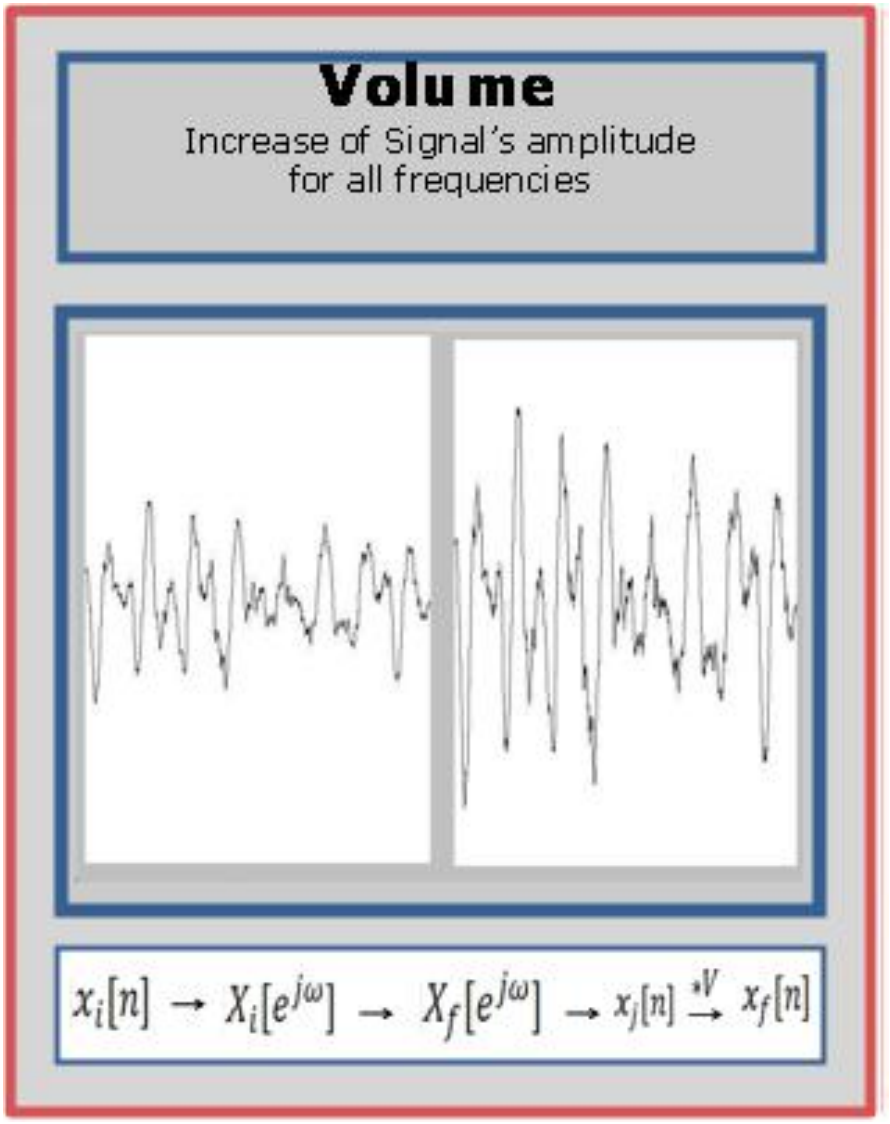

Volume
Increase of Signal's amplitude
for all frequencies
$x_i[n] \rightarrow X_i[e^{j\omega}] \rightarrow X_f[e^{j\omega}] \rightarrow x_j[n] \xrightarrow{*V} x_f[n]$

Figure 5.11

5.6 MATLAB EQ: Results[6]

5.6.1 Results

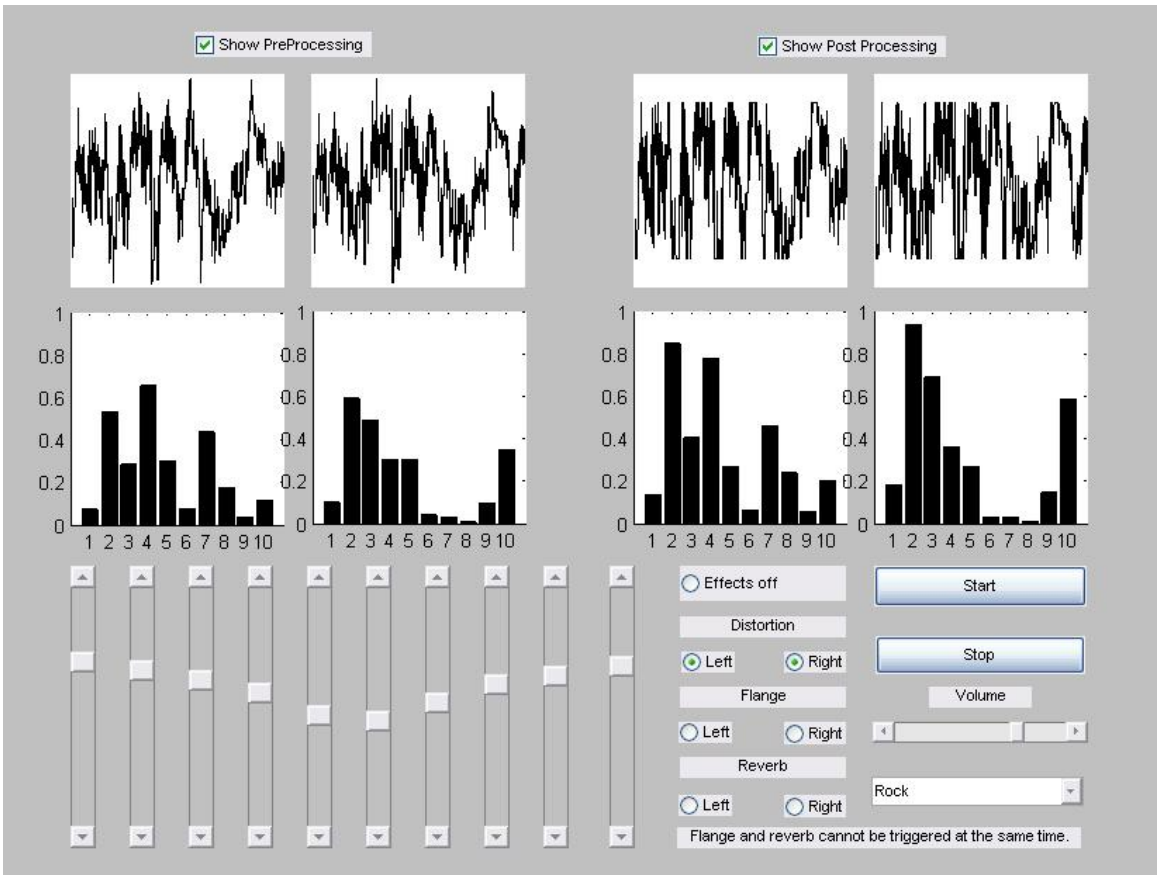


Figure 5.12

The above figure is a demonstration of the end result of our Visual Equalizer.

As shown, we were able to implement a real-time spectrum analysis of the signal in dual-channel (stereo) mode, with the preprocessed signal on the left and the post-processed signal on the right. The graphs on the bottom represent the 10 binned frequency blocks in a logarithmic queue up to 22.05 KHz.

Each frequency block can be adjusted to modify the gain levels. The Distortion, Flange, and Reverb options implement each command in the left channel, right channel, or dual-channel modes. The Volume control is the Unity Gain, increasing the amplitude over all the frequency blocks. The Start and Stop buttons begin or end the visual spectrum processing.

As a bonus, we threw in audio presets such as Rock, Treble Booster, Bass Booster, etc.

[6] This content is available online at <http://cnx.org/content/m15656/1.1/>.

5.7 MATLAB EQ: The Team[7]

5.7.1 MATLAB EQ: The Team

Just us guys.

Figure 5.13

Chris Corbet

Chris is a senior electrical engineering and physics major at Rice University. He is a very sensitive guy who enjoys watching sunsets and long walks on the beach. IN HELL.

Nik Maureira

Nik is a junior electrical engineering major at Rice University. He is the classy one of the group, and would do anything for a bottle of fine wine...WITH YOUR GIRLFRIEND.

[7] This content is available online at <http://cnx.org/content/m15653/1.2/>.

Figure 5.14

Alex Mrozack

Alex is a junior electrical engineering major at Rice University. He brings soul and style to the group, ready to dunk on you in his Nikes or his street shoes. IN YOUR FACE.

Chapter 6

CANADA: Continuous Adaptive Non-Linear Analysis for Driving Applications

6.1 Introduction[1]

6.1.1 Introduction

As personal computer applications become increasingly complex and personal computers themselves become increasingly powerful multi-process machines, the issue of application control becomes an important issue. While the keyboard and mouse previously reigned as the de facto computer inputs, the capability of today's computers to do fast, adaptive signal analysis gives way to new and exciting control opportunities.

Most computers are equipped with basic microphones. Though the idea of spoken word or sound control is not a new idea, most voice control systems require complex algorithms to filter noise and account for the many differences between peoples' voices. Our project takes advantage of the simple frequency patterns in whistles in order to simplify this process.

To handle signal processing, we chose to utilize the features of the popular LabView development environment. Combined with the powerful Java programming platform, we are able to send control commands in order to drive an application, which in our case was iTunes. In the scope of our project, LabView instantiates a Java control program which then passes commands onto iTunes.

6.2 Algorithm Overview[2]

6.2.1 Algorithm Overview

The first step in detecting a signal is to input it into the system. Since we are using an audio signal, a microphone is the obvious choice. However, the desired control signal is not the only sound in the room. There is also the music that is being played through the speakers as well as outside noise. Unfortunately, there is not much that can be done about the random noise that is present in the room. However, there are tools available that allow us to minimize the interference caused by the music. Specifically, we can use an adaptive filter to mimic the room's effect on the output of the sound card, providing us with an estimate of the music's contribution to the signal received by the microphone. We can subtract the microphone's signal from this estimate in order to obtain—hopefully—only the whistle.

[1] This content is available online at <http://cnx.org/content/m15674/1.2/>.
[2] This content is available online at <http://cnx.org/content/m15673/1.2/>.

Figure 1 (Figure 6.1: Our System's Block Diagram) shows the block diagram of our system. All of these components fall into four main categories:

1. Signal acquisition
2. Whistle isolation
3. Whistle frequency analysis
4. iTunes interface

The acquisition phase is the top portion of the diagram, the whistle isolation system is represented by the $\hat{h}$ box and the band pass filter, the rest of the diagram (except for the Java controller) comprise the whistle analysis phase, and the Java controller is the iTunes interface.

Our System's Block Diagram

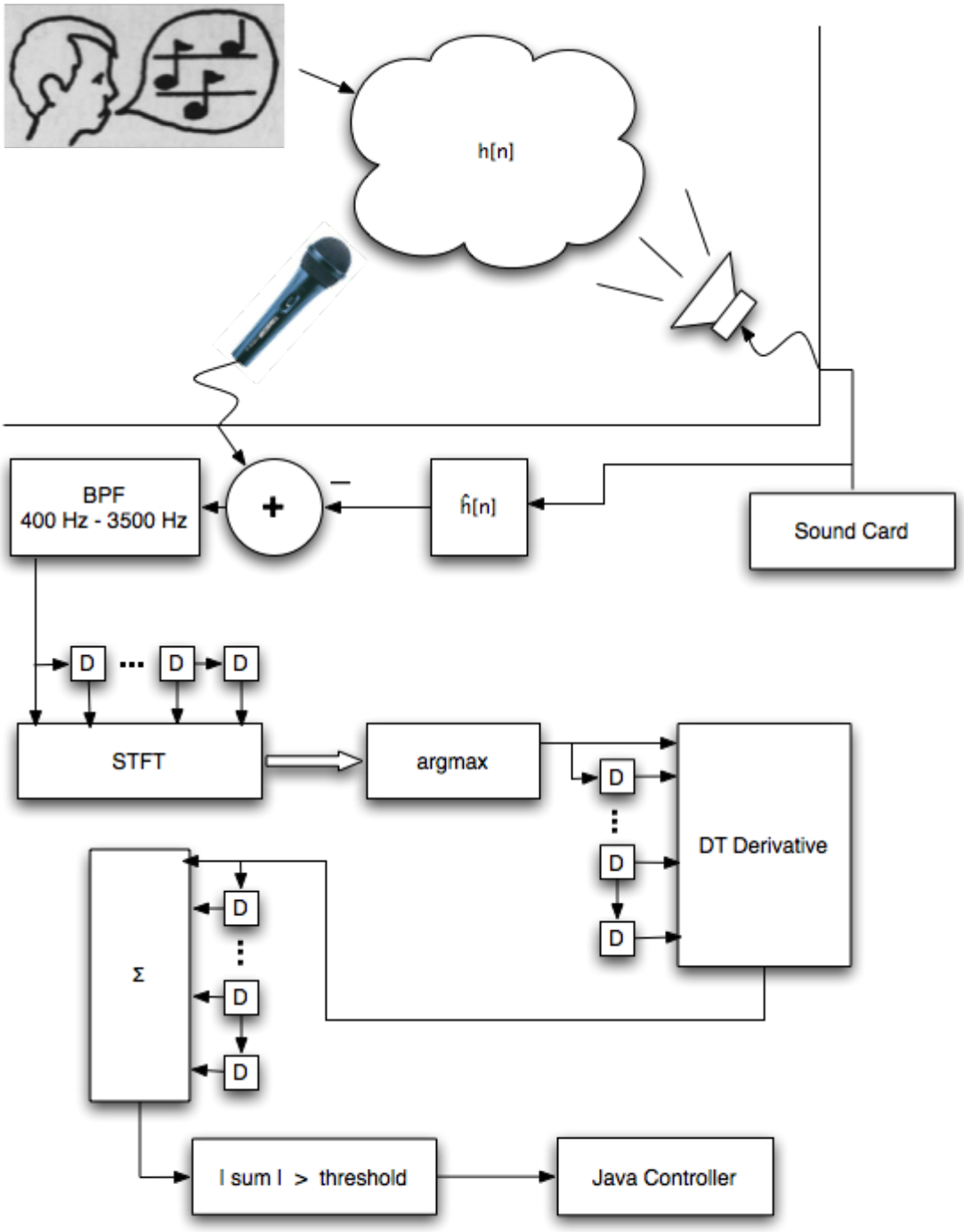


Figure 6.1: In our system, the sound is output through the speaker, and the microphone receives the music and whistles while the sound card receives the audio without the room affecting it. We then remove the music and process the whistle.

After isolating the whistle, we apply a band pass filter whose pass band corresponds to common whistle frequencies in order to remove extraneous noise outside of these whistle frequencies.

We then take the Short Time Fourier Transform in order see how the frequency components of the whistle change over time. If the frequency of the whistle is increasing iTunes should advance to the next track, and a decreasing frequency will skip to the previous track. To accomplish this, we examine the frequency with maximum power (the argmax in the figure below) and accumulate several readings of this frequency. In order to see if this function is increasing or decreasing we take the derivative and examine its average value. If the average value is positive, the function must have been increasing and the whistle must have been from high to low frequencies.

6.3 Least Mean Squares Adaptive Filters[3]

6.3.1 Least Mean Squares Adaptive Filters

6.3.1.1 Why LMS?

In our system, the music that is being played by the computer is a known output that is being fed into the room, which has an unknown response, and picked up by the microphone. Since we have the signal both before and after the influence of the room, this is a candidate for an adaptive filter. The adaptive filter can be used to try to estimate the room's response based on the music being outputted and the input to the microphone. Once an estimate is available, we can use it to remove the interference of the music from the input to the microphone.

6.3.1.2 LMS Overview

One of the most popular adaptive algorithms used today is the Least Mean Squares (LMS) algorithm. Essentially, this algorithm attempts to minimize the error that occurs between the detected (or desired signal), d[n], and the estimated value of the signal, y[n]. This estimated signal is created by taking the original input, x[n] , and running it through an approximation of the unknown channel. Basically:

$y[n] = x[n] * \overset{\Theta}{h}[0] + x[n-1] * \overset{\Theta}{h}[1] + \cdots + x[n-N] * \overset{\Theta}{h}[N]$

or

$y[n] = x_N^T[n] * \overset{\Theta}{h}[n]$

where N is the order to which you are approximating the unknown system, x_N is a vector of the last N value of x, and each h[n] is a weight.

[3]This content is available online at <http://cnx.org/content/m15675/1.2/>.

LMS Algorithm Block Diagram

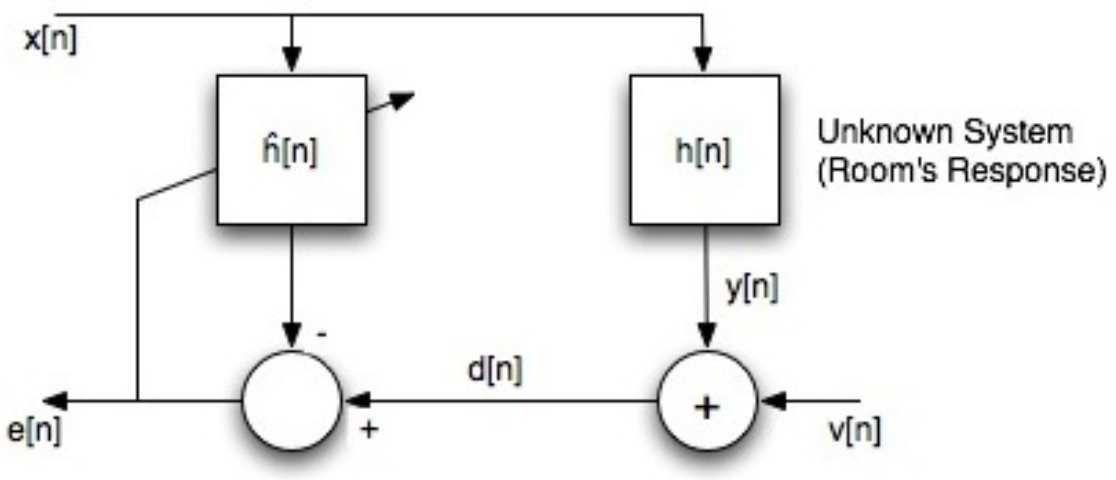


Figure 6.2: x[n] is the input to the filter, v[n] is the interference in the room, h[n] is the impulse response we're modeling, and e[n] is the error in this modeling.

Once the signal and its approximation are found, the error is just the difference between the two signals at the current point in time.

$e[n] = d[n] - y[n]$

Using the error, we can approximate the next set of weights as follows:

$\overset{\Theta}{h}[n+1] = h[n] + \mu * e[n] * x_N[n]$

where μ is a constant.

6.3.1.3 Choosing μ

The choice of μ is an important one as it greatly affects how the system will perform. Small values allow for greater precision in convergence, but slow down the response time of the system. If the value is too small, then the adaptive filter will not adapt fast enough. If it is too large, the system will not converge at all, and the value weights will actually diverge. Experimentation with different systems and environments is necessary to choose the ideal value for μ.

6.3.1.4 Isolating the Whistle

It can be shown that the error value will converge on a minimum point. This theoretical minimum will likely not be at zero error, since there is outside noise and interference, but it represents the original signal's effect on d[n] being completely removed. In the case of our signal detection system, the whistle that we are trying to detect will still be in the e[n] and therefore we will use e[n] as the input to the rest of our system. Once the adaptive filter has settled down, this signal will be the input to the microphone minus the estimated value of the music when it reaches the microphone.

6.4 Joint Time-Frequency Analysis[4]

6.4.1 Joint Time-Frequency Analysis

6.4.1.1 Short Time Fourier Transform

After our system has isolated the whistle sound, it is input to a continuously running analysis algorithm based on the result of a Short-Time Fourier Transform. While the traditional Fourier transforms do not maintain a sense of time, the STFT allows for joint time-frequency analysis. The formula for the STFT is:

$\mathrm{STFT}\{x[n]\} = X(m,w) = \sum_{n=-\infty}^{\infty} x[n]\, w[n-m]\, e^{-\mathrm{jwn}}$

Individual sections of the signal are windowed and their FFT is taken. The result of this operation is a two dimensional function in terms of the offset from zero ("time") and the frequency content of the signal. According to the Heisenberg Uncertainty Principle, however, we cannot have an arbitrary amount of resolution in both the time and frequency domains. For our application, we are only interested in detecting trends in the frequency with highest power, so time resolution is less important.

6.4.1.2 Spectrogram

Squaring the magnitude of the STFT results in a 3-D function is known as a spectrogram. The spectrogram of a whistle whose pitch goes from low to high looks like this.

[4]This content is available online at <http://cnx.org/content/m15676/1.2/>.

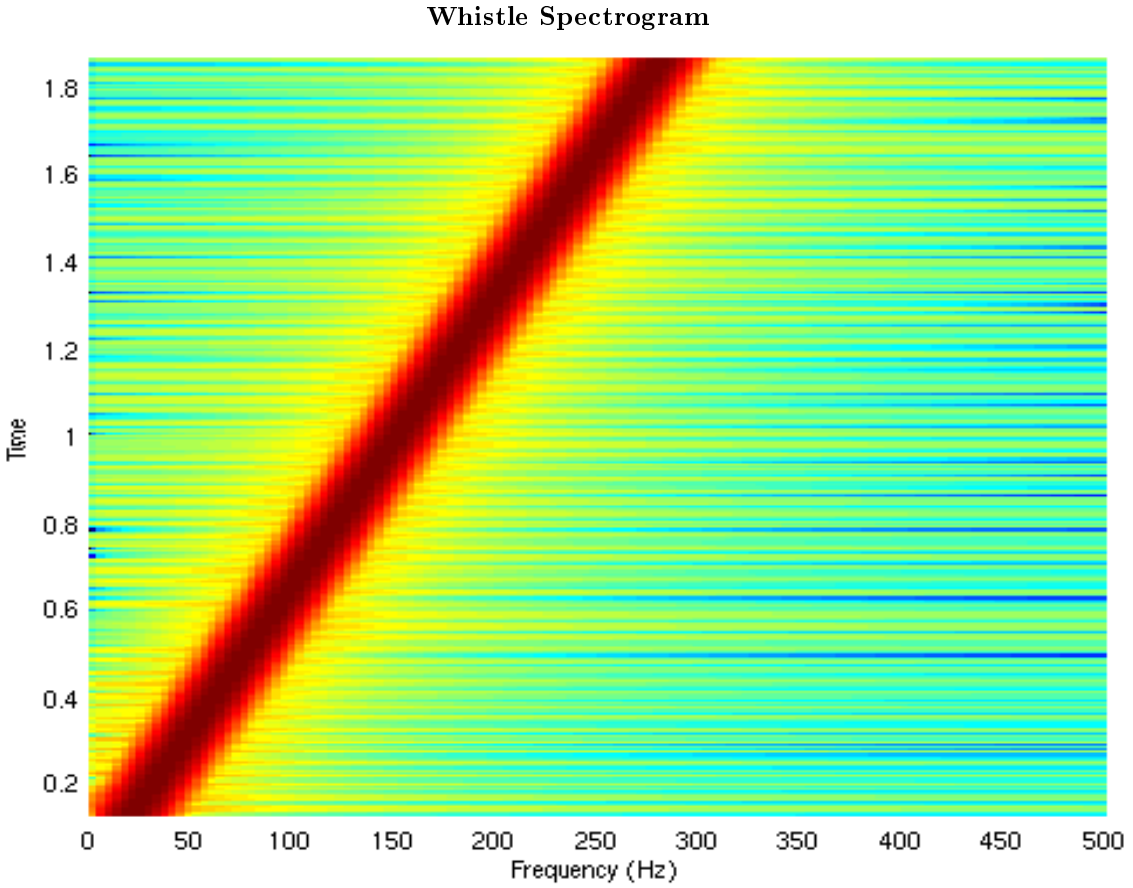


Figure 6.3: This is a whistle of increasing frequency. Each slice of the spectrogram is an FFT of a certain windowed chunk of the signal. This embeds a sense of time in the spectrogram.

Notice that there is a dominant power (dark red) that shows a very clear upward trend over time. Each slice of the frequency axis at a particular time corresponds to a single chunk of the signals' FFTs. The dark red corresponds to the maximum of one of these FFTs, and as time passes (in the positive vertical direction), we see the frequency of highest power increases. Below is a graph of three of these component FFTs, illustrating how the peak frequency increases across time. Since pitch and frequency are essentially synonymous, we can determine what kind of whistle is input by looking at trends in the frequency with the dominant power.

Power vs. Frequency for 3 Spectrogram Components

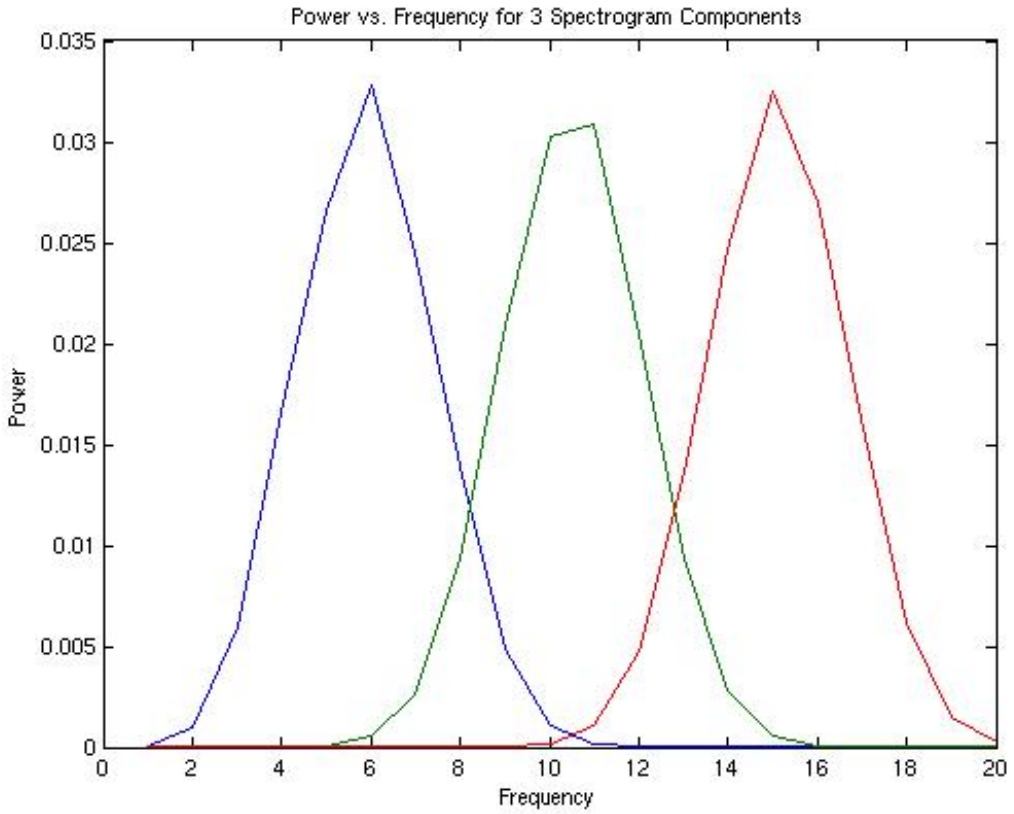


Figure 6.4: Here are 3 of the FFTs that comprise the spectrogram above. The frequency of maximum power is increasing.

With the dominant frequency for any given windowed input now known, our system then takes the discrete-time derivative of that frequency over the window. Calculus tells us that the derivative of a function at a point is positive for increasing functions and negative for decreasing functions. By continuously taking derivatives of these windows our system can track the basic shape of the spectrogram.

6.4.1.3 Analysis using the STFT

Our analysis algorithm keeps a running buffer of the signs of these discrete-time derivatives. It then takes the magnitude signs inside the buffer. If the buffer encounters a number of continuous signs above or below a certain threshold, either positive or negative, it concludes that the input is a whistle with increasing or decreasing pitch, respectively. Finding the optimal threshold value is mostly trial and error; we looked at recordings of sine waves with constant frequencies and white noise in order to determine a reasonable upper bound for the area under the derivative curve.

There is a tradeoff between the size of the buffer and the quality of the analysis. Given that a whistle may last up to three seconds, it's clear that the buffer need not contain that many samples in order to find

and characterize the whistle. But on the same token, too few samples will result in an unusual number of false positives and change the song without any user interaction. And while this is not necessarily complex math, doing it quickly and continuously requires the window be as small as possible.

6.5 iTunes Control Interface[5]

6.5.1 iTunes Control Interface

To accomplish anything useful with our signal processing, we need to drive an application. We chose iTunes, a popular multiplatform program for playing digital music files, as our example application for several reasons. First, the ability to control iTunes through external means is well documented and fairly simple to achieve. More important, however, is the aspect of music or audio playback in a room while simultaneously listening for whistles.

6.5.1.1 Java-COM

iTunes is both the system to control as well as the dominant source of noise to combat. Based on the profile of the whistle detected, whether it be increasing or decreasing in frequency, we will skip to the next or previous track in iTunes. To accomplish this, LabVIEW makes a call to a Java program that interacts with iTunes by accessing the Component Object Model (COM) interface of Windows. The JACOB[6] library serves as an intermediary to complete this task. JACOB[7] is a native Java-COM bridge, allowing Java programs to simultaneously run and interact with system applications. Once the COM interface receives instructions from LabVIEW, it then interacts with iTunes and changes tracks appropriately.

6.6 Conclusion and Possible Improvements[8]

6.6.1 Conclusions and Possible Improvements

While we did not create a fully working prototype, we believe that our algorithm is a solid approach for implementing a solution to this problem. While trying to implement the whistle detection in LabVIEW, we were faced with obtaining synchronized samples from two audio input sources while outputting through another. In order to assure that all samples are read in at the same time, we would need to use real time operating system or at least the real time LabVIEW plugin. We believe the best solution to this problem would be to implement the system in hardware where we would have complete control over sample acquisition.

We were able to implement the LMS filter in MATLAB, but its running time was too long to be a feasible solution. Its running time appeared to be O(n^2) and running the filter on a 10 second clip from a song took 4 hours to process. Again, since the LMS filter is such a popular adaptive filter, more efficient implementations must exist; and implementing it in hardware would surely provide a significant performance improvement.

Finally, one of the most important advantages of our design is that it's modular. While we chose to implement a rising/falling frequency detector, any type of aural recognition could be implemented with this system. For example, we could replace the STFT and increasing/decreasing detector with a voice recognition system.

[5] This content is available online at <http://cnx.org/content/m15677/1.2/>.
[6] http://danadler.com/jacob/
[7] http://danadler.com/jacob/
[8] This content is available online at <http://cnx.org/content/m15678/1.2/>.

Chapter 7

Tomographic Processing of Spotlight-Mode SAR

7.1 Introduction[1]

7.1.1 Introduction

Our project covers an investigation of Synthetic Aperture Radar (SAR) and the Matlab processing of SAR data we received from Ohio State University. The data was generated by a simulation of spotlight-mode SAR, a specific type of SAR which takes advantage of a moving sensor tracking a single ground target. To process this data, we utilize digital signal processing techniques such as interpolation and windowing along with a knowledge of how SAR works.

7.2 Background[2]

7.2.1 Synthetic Aperture Radar: Background

7.2.1.1 What is Synthetic Aperture Radar?

Synthetic Aperture Radar (SAR) is a microwave imaging system that is used to obtain high resolution pictures of large areas of terrain. The radar can be either airborne or spaceborne. As the platform moves, closely spaced pulses are transmitted and the reflected signals are received and processed using Fourier methods. The processed data resembles data taken with a system that has a very large antenna, thus allowing extremely high resolution.

Synthetic Aperture Radar was first developed in the early 1950's. The earliest type of SAR is called strip-mapping mode SAR. It is primarily used for imaging large areas of terrain, such as the surface of a nearby planet. This mode emits the radar pulses at a constant "look" angle to the surface while traveling along a flight path or orbit. This process creates a strip of mapped ground, and can be repeated along a polar orbit to map the entire surface of a planet.

Spotlight mode SAR is a newer form of SAR and was developed in the early 1980's. It is more widely used today than strip-mapping mode and it is what our project deals with. In spotlight mode, the radar is steered continually as the carrier of the radar flies over a patch of ground. In another word, the "look" angle is constantly adjusted so that a single patch of ground is always illuminated. This method allows for higher resolution in the azimuth "travel" direction of the platform but is not able to image as large of an area as strip-mapping mode.

[1] This content is available online at <http://cnx.org/content/m15658/1.1/>.
[2] This content is available online at <http://cnx.org/content/m15667/1.1/>.

7.2.1.2 How Does Imaging Radar Work?

As mentioned earlier, we use the Synthetic Aperture Radar processing technique because of its advantages when it comes to imaging large areas at high resolutions. However, why do we even use radar to image things in the first place?

Radar is used in imaging because of the minimal constraints that is has on time-of-day and atmospheric conditions. The area of imaging does not have to be illuminated by sunlight in order to obtain a picture. This allows for continuous mapping regardless of the position of the sun, which saves time and therefore, money.

Radar also has the ability to penetrate cloud cover because one can choose a wavelength that is not absorbed by water. This fact is what allowed scientists at NASA to provide stunning images of the surface of Venus, which is completely shrouded in cloud-cover.

Imaging radar works by emitting a signal and then recording the strength of the reflected signal (scattering coefficient) for that area. The pulses are emitted at an angle to the surface such that if they strike a smooth, flat surface, very little of the signal will be reflected back towards the antenna which corresponds to a darker spot on our scattering coefficient image.

When the radar pulse strikes uneven surfaces such as urban areas or vegetated areas, the signal gets reflected numerous times and there is an increased likelihood that the radar antenna will eventually receive a large portion of the signal back, corresponding to a whiter spot on your image. Scientists use this fact to determine the extent of flooding in urban areas or to discern how much an oil spill in the ocean has grown.

7.2.1.3 Synthetic Aperture Radar and Microwave Imaging System

The resolution of an image taken from an imaging system is usually determined by the size of the Aperture (lens for optical systems and antenna for radar). Conventional radar systems use passive methods deployed with optical or short-wave infrared sector that rely on sunlight reflection. On the other hand, synthetic Aperture Radar uses a microwave imaging system. Two important advantages resulting from using microwave pulses are that cloud cover can be penetrated and the imaging process can be performed at night.

However, antenna size limits one from applying microwave imaging systems. A very large size of antenna is required to obtain satisfactory resolution. Therefore, the size of the antenna makes it impractical for the radar carrier.

Synthetic Aperture Radar solves the problem by "simulating" a large Aperture. The radar sends and receives signals from a relatively small antenna while the platform traveling along a flight path. One can then use the digital signal processing techniques to combine the data into a coherent image. The result is the same as if one has used a very large antenna.

7.3 Projection-Slice Theorem[3]

[3] This content is available online at <http://cnx.org/content/m15663/1.1/>.

7.3.1 Brief Review of Computer-Aided Tomography

Computer-Aided Tomography, or CAT (as in CAT scan) is a technique for remote 2-D and 3-D imaging. By moving a sensor around a target, one can collect sufficient 1-dimensional data to reconstruct the original multidimensional image. This process utilizes an amazing relationship called the Projection-Slice Theorem, which states that each piece of projection data at some angle is the same as the Fourier transform of the multidimensional object at that angle. Using a range of data from a range of angles, one can, given sufficient computation resources, reconstruct the actual image by taking the inverse transform. The Projection-Slice Theorem has found a range of applications in remote sensing, the most famous of which is the 3-D imaging of humans, popularly known as the CAT scan. The focus of this project, Spotlight-Mode Synthetic Aperture Radar, uses the Projection Slice Theorem in a way quite similar to CAT scan technology, except the way radar projections are generated by the image is slightly different from the way CAT scans use X-rays.

7.3.1.1 Projection-Slice Theorem

Let g(x,y) represent the radar reflection of our image. The two-dimensional Fourier transform of g is defined as

$$G(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x,y)e^{-j(xX+yY)}\,dx\,dy$$

Figure 7.1

And

$$g(x,y) = \frac{1}{4\pi^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} G(x,y)e^{j(xX+yY)}\,dXdY$$

Figure 7.2

We can model the reflection behavior of the incident radar by considering the following overhead diagram

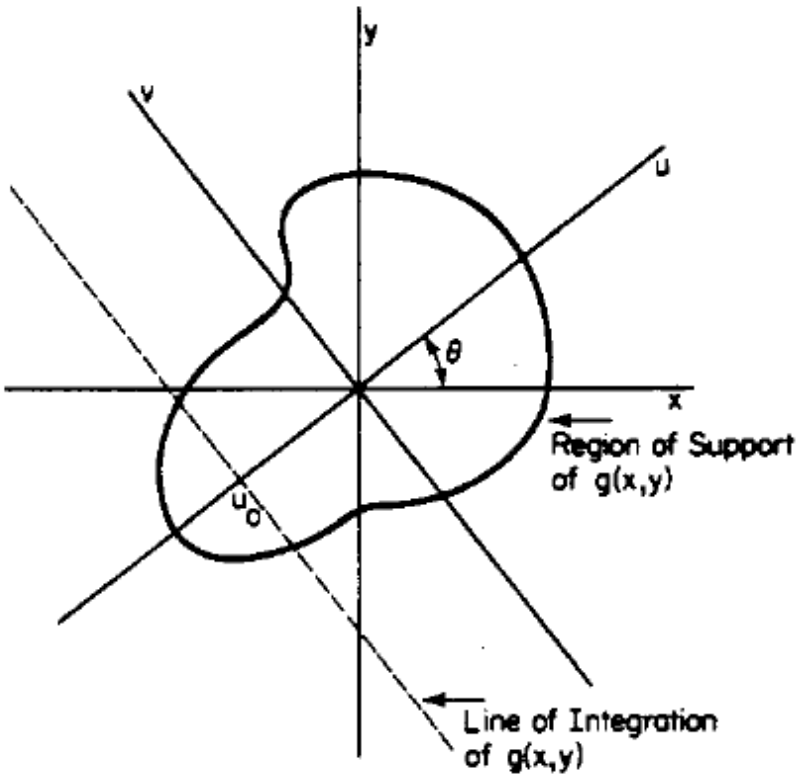


Figure 7.3

The smooth line outlines our image g(x,y), and the horizontal and vertical axes x,y are overlaid. The radar is incident upon the target along the axis of the path **u** at an angle theta. For a target which is far away, the radar wave front is approximately flat, and so this means that a reflected beam which has traveled a certain unique distance to and from the sensor comes from a straight path across the image, perpendicular to **u**. This path is in the direction of **v** and can be represented by a line integral in the direction of **v** at position **u0**. The formula for this is given by

$$p_\theta(u) = \int_{-\infty}^{\infty} g(u\cos\theta - v\sin\theta, u\sin\theta + v\cos\theta)\, dv$$

Figure 7.4

The 1-D Fourier transform of **p(u)** is given by

$$P_\theta(U) = \int_{-\infty}^{\infty} p_\theta(u) e^{-juU}\, du$$

Figure 7.5

And then, through applying the equation for **p(u)** and simplifying, we are left with

$$P_\theta(U) = G(U\cos\theta\,, U\sin\theta)$$

Figure 7.6

This is the Projection Slice Theorem! What this states is that the Fourier transform of a projection taken at an angle theta is **equal to** the 2-D Fourier transform of the image at that same angle theta. To reconstruct the original image, one must merely take the inverse Fourier transform in two dimensions of a set of data **P(U)**. This is not as easy as it sounds for reasons discussed later. Notice how the Fourier transform of the image **G** does not have the usual form, **G(X,Y)**. It is instead expressed in polar form, and the variable theta lets us know that we have only a slice of the transform for each P(U).

7.4 Tomographic Processing[4]

7.4.1 Data Processing

In the words of the highly esteemed Rich Baraniuk, the signals received by the radar sensor must be "munjed" upon in order that the user can learn anything useful at all. We flesh-out the basic spotlight-mode SAR derivation from start to finish, noting the places in which we make approximations, all the while aiming at interpreting our bit stream into the meaningful pieces of the Projection-Slice Theorem. Something to note is that this theoretical approach does not include any Doppler shift analysis. Other approaches to synthetic aperture radar heavily rely on phase data collected during a physical flyby of the target, where instrument velocity plays an important role. The mathematics in this section follows as in David Munson's 1983 paper on "A Tomographic Formulation of Spotlight-Mode Synthetic Aperture Radar."

7.4.1.1 The Setup

The way spotlight-mode SAR collects data samples is by gathering image projections from a range of angles. In our case, this range is broken down into a set of equally spaced angles so that essentially we have snapshots at various views around a target. A depiction of what it would look like is given below.

[4]This content is available online at <http://cnx.org/content/m15661/1.1/>.

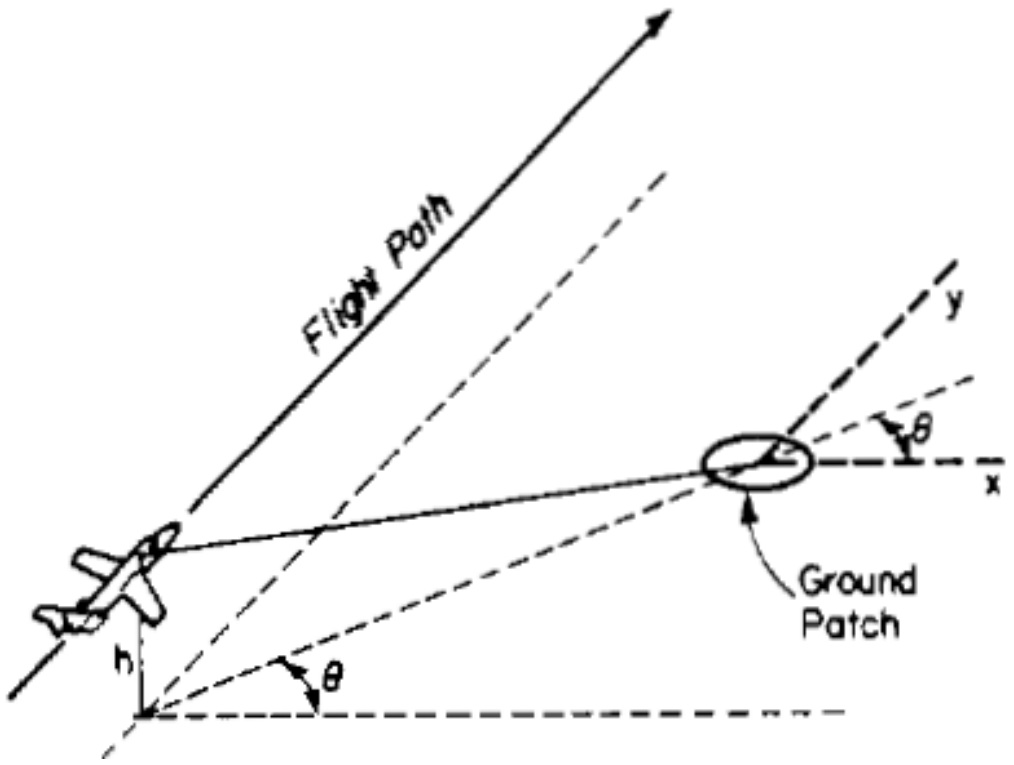


Figure 7.7

This drawing shows how the altitude of the sensor platform might play a role in the angular view of the target. For our derivation, we will ignore this parameter and assume that the radar is somehow incident at ground level, that as the sensor moves closer to the target distances remain undistorted by this variation in 3-dimensions. This ground plane geometry is as shown below.

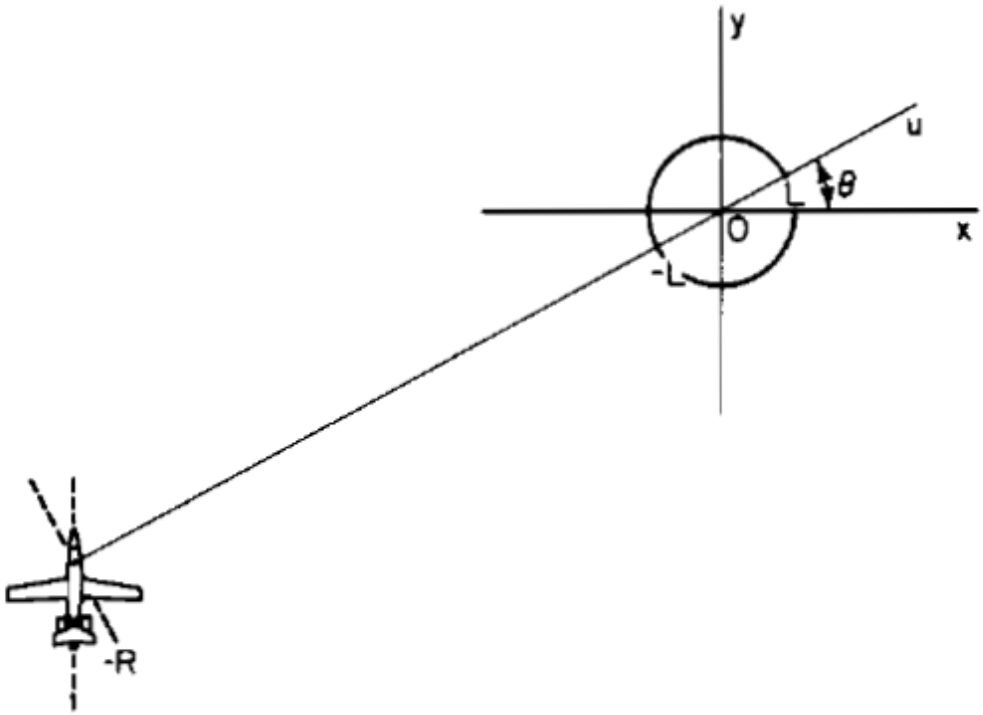


Figure 7.8

Note that the angle **theta** is the same as that in the description of the Projection-Slice Theorem. The distance from the center of the target image is given by the variable **R**, and the radius of a circular target is given by **L**. The radar signal travels along and parallel to path **u**.

7.4.1.2 Generating the Reflection Signal

At this point we are ready to start generating our signals! Our radar device works in a microwave frequency range designed to penetrate clouds and other obstructions with ease. It emits a linear FM chirp pulse waveform **Re{s(t)}** where

$$s(t) = \begin{cases} e^{j(\omega_0 t + at^2)}, |t| \leq \frac{T}{2} \\ 0, \quad otherwise \end{cases}$$

Figure 7.9

In this signal, **w0** is the RF carrier frequency and **2a** is the FM rate. The frequency rises linearly with time so that the minimum frequency is **w0-aT** and the maximum is **w0+aT**. The point reflection off of a reflection coefficient at **(x0,y0)** given by **g(x0,y0)** can be written

$$r_0(t) = A \cdot Re\left\{g(x_0, y_0)s(t - \frac{2R_0}{c})\right\}dxdy$$

Figure 7.10

Where **R0** is the distance of **g(x0,y0)** from the radar, A accounts for propagation attenuation, **c** is the speed of light, and **2R0/c** accounts for the two-way travel time from radar to target.

7.4.1.3 Interpreting the Reflection Signal

Points on the target ground patch equidistant from the radar lie on an arc, but typically **R≫L**, so this arc is nearly, and may be approximated as, a straight line. Combining this approximation with the polar formulation of a differential line of scatterers (radar reflectors), we can write down a new relation involving a polar representation of the reflection.

$$r_1(t) = A \cdot Re\left\{p_\theta(u_0)s\left(t - \frac{2(R + u_0)}{c}\right)\right\}du$$

Figure 7.11

If **R≫L**, we may take **A** as a constant over **u**, and this enables us to write the reflection from the whole ground patch as the integral of **r1** over **u**.

$$r_\theta(t) = A \cdot Re\left\{\int_{-L}^{L} p_\theta(u)s\left(t - \frac{2(R + u)}{c}\right)du\right\}$$

Figure 7.12

This integral has the form of a convolution! This provides us a good hint that Fourier methods might be the right way to analyze this signal. The signal stated here is really the raw data that we receive from the radar. The return chirp is the projection slice convolved with the initial pulse.

7.4.1.4 Mixing the Reflection Signal

It turns out that the correct way to process this raw data is to mix it with the starting signal, **s(t)**. Written out, **r(t)** has the form

$$r_\theta(t) = A \cdot Re\left\{\int_{-L}^{L} p_\theta(u) \exp\left\{j\left[\omega_0\left(t - \frac{2(R+u)}{c}\right) + a\left(t - \frac{2(R+u)}{c}\right)^2\right]\right\} du\right\}$$

Figure 7.13

$$-\frac{T}{2} + \frac{2(R+L)}{c} \leq t \leq \frac{T}{2} + \frac{2(R-L)}{c}$$

Figure 7.14

Mixing this signal with the real and imaginary parts of the signal **s(t)**, low-pass filtering the two, and then adding them together gives us a complex signal.

$$C_\theta(t) = \frac{A}{2}\int_{-L}^{L} p_\theta(u) \exp\left\{j\frac{4au^2}{c^2}\right\} \cdot \exp\left\{-j\frac{2}{c}\left(\omega_0 + 2a\left(t - \frac{2R}{c}\right)\right)u\right\} du$$

Figure 7.15

The quadratic term in the exponential can be approximated as 0, and as that term disappears, we get a very profound result

$$\overline{C}_\theta(t) = \frac{A}{2}\int_{-L}^{L} p_\theta(u) \exp\left\{-j\frac{2}{c}\left(\omega_0 + 2a\left(t - \frac{2R}{c}\right)\right)u\right\} du$$

Figure 7.16

Which is the Fourier transform of **p(t)**.

$$\bar{C}_\theta(t) = \frac{A}{2} P_\theta \left[\frac{2}{c} \left(\omega_0 + 2a \left(t - \frac{2R}{c} \right) \right) \right]$$

Figure 7.17

7.4.1.5 Interpreting Our Result

From our formulation of **r(t)** above, we know the restriction on **t**. If we consider the argument of **P( )** to be **X**, the radial spatial frequency, we know **P(X)** is only determined for **X** between **X1** and **X2** where

$$X_1 = \frac{2}{c} \left(\omega_0 - aT + \frac{4aL}{c} \right)$$

Figure 7.18

$$X_2 = \frac{2}{c} \left(\omega_0 + aT - \frac{4aL}{c} \right)$$

Figure 7.19

The term **4aL/c** will be negligible for typical SAR, so we can see that **X1** and **X2** are proportional to the lowest and highest frequencies in the transmitted chirp pulse. **X1** and **X2** correspond to the inner and outer radii for which **P(X)** is defined.

$$\frac{2}{c}(\omega_0 - aT) \leq X \leq \frac{2}{c}(\omega_0 + aT)$$

Figure 7.20

C(t) is the final form of the processed data! What this tells us is that after mixing the reflection with the real and imaginary parts of the original chirped pulse, low-pass filtering, and linearly combining the two, we are left with the Fourier transform of the projection **p(t)**. From here we need to take the inverse transform to finally reconstruct **g(x,y)**. Unfortunately, we have the polar form of a Fourier transform, whose known values would be located somewhere on this Locus

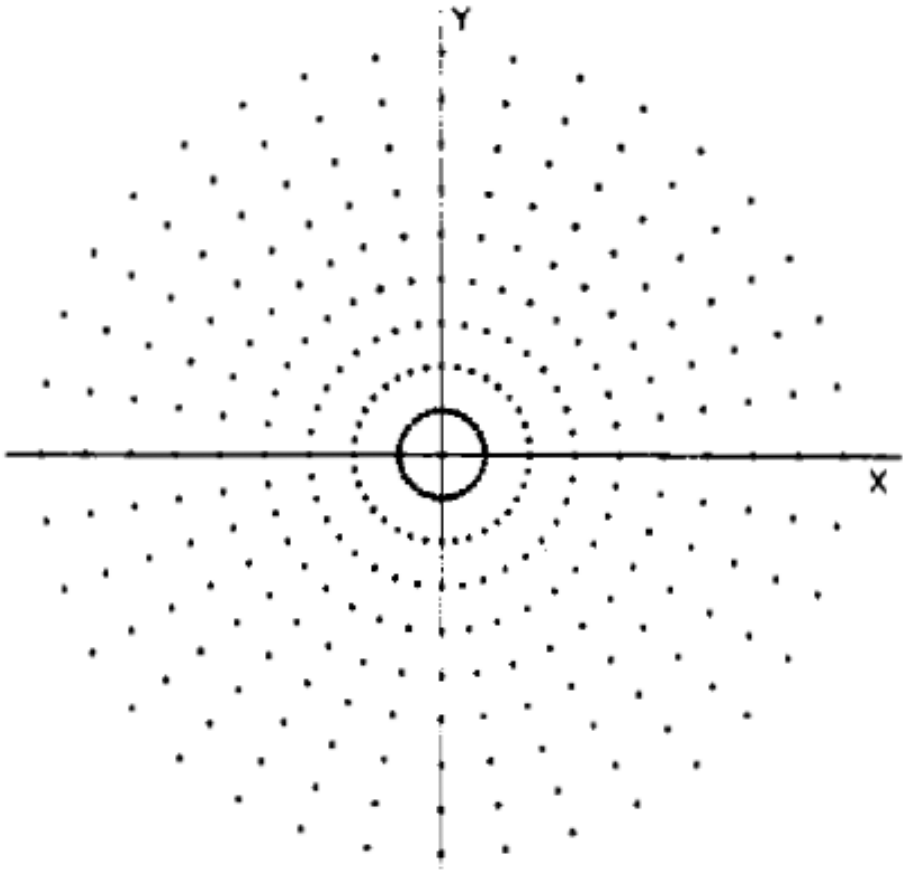


Figure 7.21

There exists a 2-D polar inverse Fourier transform, and this is the preferred method of transform for the optimal quality reconstruction. This method, called Convolution Back-Projection, requires an enormous number of computations to execute, however, and so in practice one has to either have a quite powerful system or be ready to wait for results. The practical method attempts to remedy this issue by utilizing the Fast Fourier Transform, or FFT. Although there is no known 2-dimensional FFT or inverse FFT for polar coordinates, one may interpolate the polar data to rectify the coordinate system and then apply the 2-D inverse FFT in the regular fashion. The rectification process requires windowing the data, and interpolation is by its nature inexact, so this method ends up trading a great deal of resolution for extra speed.

7.5 Data and Processing[5]

7.5.1 Introduction and Preparation of SAR Data

In order to simulate the processing of SAR data, we received SAR data from the ECE department at Ohio State University. The data they gave us was acquired through a computer simulated fly-by past a CAD model of a backhoe.

Figure 7.22

The data we received from OSU was in digital format, meaning the analog mixing and low pass filtering was already completed and we received digitized versions of Cθ(t). This function was shown to be equivalent to Pθ(U), the Fourier transform of our projection slices pθ(u). The matlab file that contained this information was a 512x1541 matrix iq_lin, a vector of various Pθ signals for different values of θ, the viewing angle. There were 1541 different viewing angles, listed in az_lin, that stepped by $1/14^\circ$ for each element and varied from -10 $^\circ$ to 100 $^\circ$, a median viewing angle of 45 $^\circ$. We also received a vector of length 512 called f that contained the microwave frequencies (7-13 GHz) that were transmitted and received. By using the wideband approximation, we made the transformation from time frequency to spatial frequency via

$R \approx 2\omega/c = (4\pi/c)\, f$

where R is the radial spatial frequency and f is the microwave frequency content. By the projection slice theorem, we have that the various Pθ are arranged radially in a polar grid along the various angles θ. We then get that our data lies on a domain

$-10^\circ \leq \theta \leq 100^\circ$

$R \in \Delta\mathrm{R} = [(4\pi/c)\, f_{\min}, (4\pi/c)\, f_{\max}]$

[5]This content is available online at <http://cnx.org/content/m15659/1.1/>.

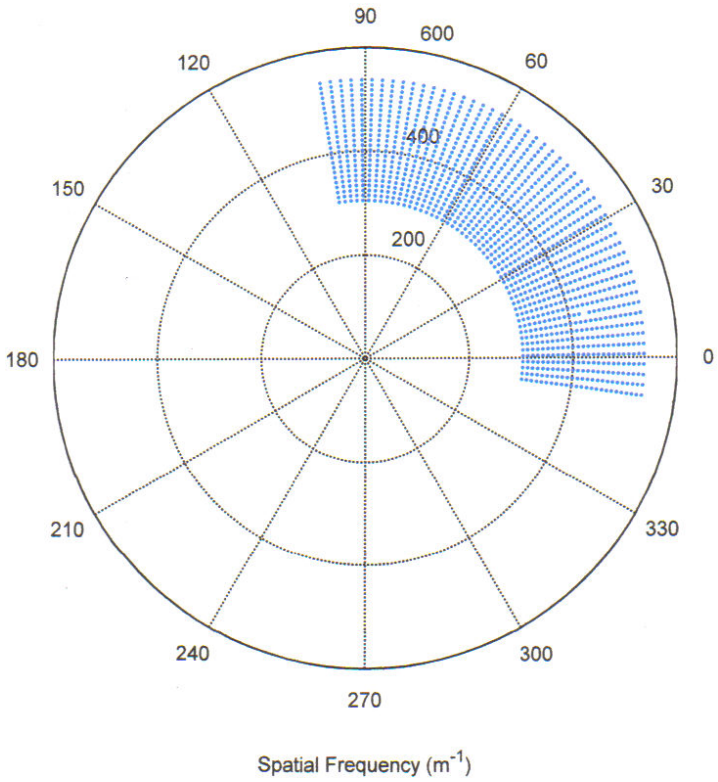


Figure 7.23

7.5.2 Processing of SAR Data

Knowing that our data is the Fourier transform of our image, after the proper preparation we want to take the inverse Fourier transform. To do this simply and efficiently (we don't want Matlab running for hours!) we linearly interpolate the data to a Cartesian grid. This is done in our Matlab function sar_lin (code found in appendix). The idea is to find an inscribed rectangular grid inside our polar data. We chose to use the square centered at 45° inscribed in our ribbon. To interpolate we made a Cartesian grid at this location and computed the polar representation of each point in order to find its 4 nearest polar neighbors. Once those neighbors were found, the Cartesian point's value was determined by linearly interpolating in the R-direction for the two θ values and then linearly interpolating in the θ-direction. The end result of our program's running of this is shown below.

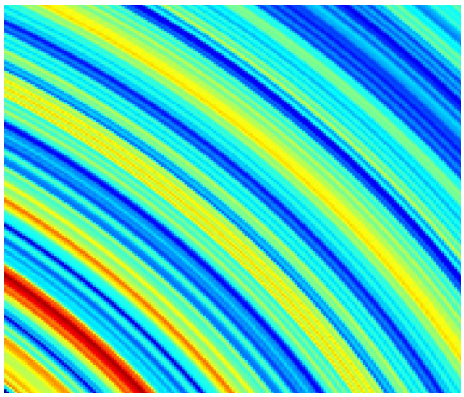


Figure 7.24

After linearly interpolating each point in the Cartesian grid we have formed, we now have our data in a form that allows us to take the 2-d inverse DFT by the fast Fourier transform method. This is what saved us computation time (program ran in about 15 seconds) and is the reason we interpolated to Cartesian coordinates to begin with. Below is the image after taking the inverse Fourier transform.

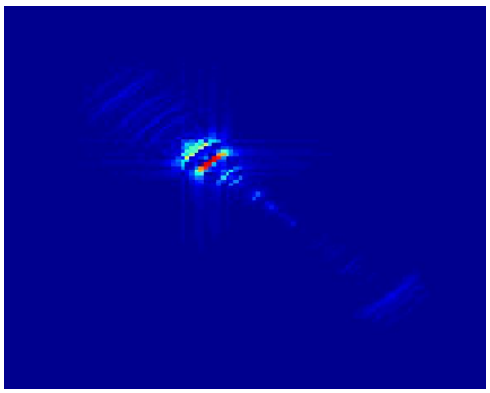


Figure 7.25

7.6 Conclusions and Future Work[6]

7.6.1 Conclusions

Our project shows that an image can be extracted from scattering coefficients obtained from SAR techniques. One need only apply some digital signal processing techniques, such as the polar to Cartesian interpolation

[6] This content is available online at <http://cnx.org/content/m15660/1.1/>.

of our data. SAR has many benefits and real world applications. For example, the microwave radiation used in SAR penetrates cloud cover. This fact has been used to image Venus' surface, providing stunning visuals otherwise unavailable. Another benefit is that SAR does not require the construction of a large antenna as the motion of the detector creates an artificial aperture.

7.6.2 Future Work

One negative aspect of our data is that the range of frequencies represented is very narrow. That is, the interpolated grid we chose focuses on a small range of frequencies relative to the amount of data that we received to work with. One potential fix for this is to do multiple interpolations of multiple Cartesian grids, ensuring no overlap, and superimpose the images that result. As long as the relative location of the grids is taken into account, then by linearity, we should be able to acquire a more detailed image by superimposing the images. One might imagine, for example, that we could superimpose the grids shown below.

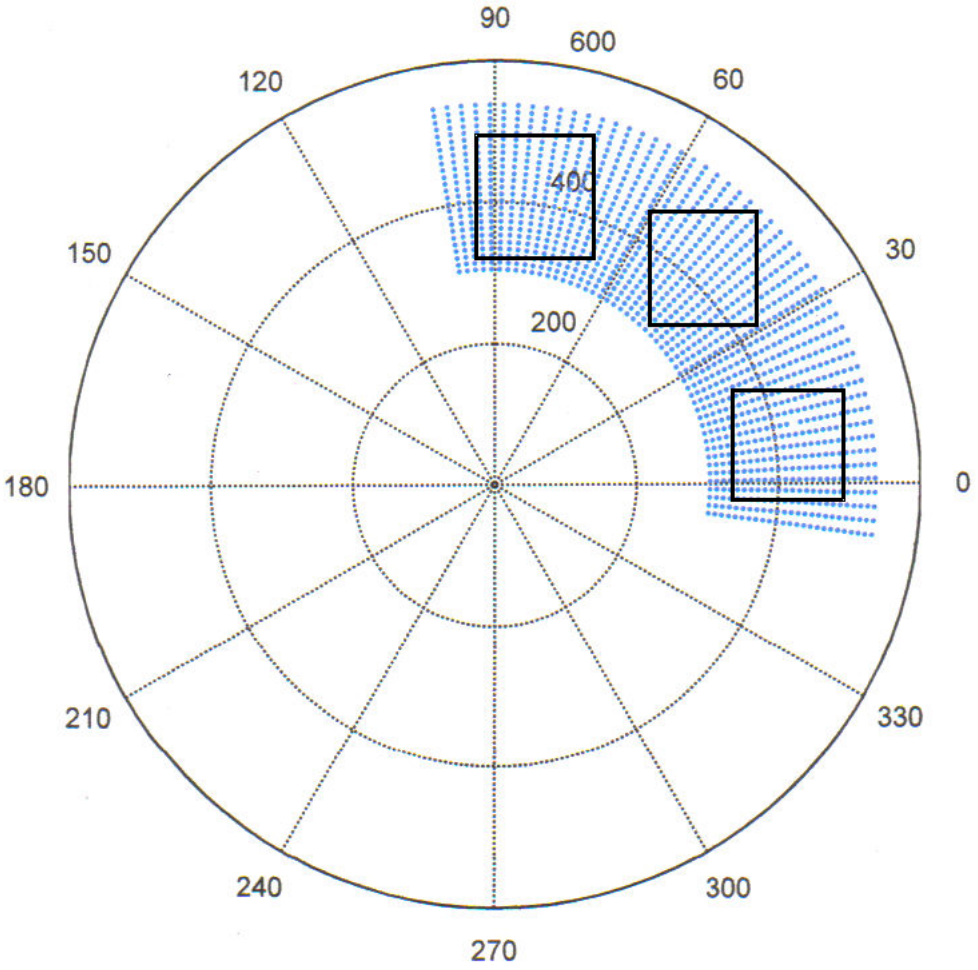


Figure 7.26

Another possibility is to forego the interpolation, which restricts the amount of data that we can use, and instead we can work in polar coordinates. This would allow us to use all of our data, cutting out nothing. The downside to this is that there is no fast Fourier transform method for polar coordinates, so the computational complexity increases the run time to be on the order of hours.

7.7 Appendix and References[7]

7.7.1 Appendix

7.7.1.1 MATLAB Function for Processing SAR Data

```
function img = sar_lin(f,az_lin,iq_lin)


% INITIALIZATION

R=4*pi*f/3e8;% Transformation of Time Freqeuncy to Spatial Frequency
A=pi/180*az_lin;% Transformation of Angle in degree to radians
for j=1:147
A(j)=A(j)-2*pi;% Adjustment so that all angular values are between 0 and pi
end

% Initialization of Cartesian Grid
X=zeros(1,175);
Y=zeros(175,1);
for j=1:175
X(j)=208+(j-1);
Y(j)=208+(j-1);
end

val=zeros(175);% Initialize the matrix of values in the cartesian grid.




% INTERPOLATOR

for j=1:175

j % Update on how far we are.

for k=1:175

r=sqrt(X(j)^2+Y(k)^2);
a=atan(Y(k)/X(k));
```

[7] This content is available online at <http://cnx.org/content/m15665/1.1/>.

```
   if (r>R(1) & r<R(512))

   % Data index of cartesian point
   J=ceil((r-R(1))*511/(R(512)-R(1)));
   K=ceil((a-A(1))*1540/(A(1541)-A(1)));

   % Linearly interpolate the cartesian point with its 4 nearest neighbor polar data
points.

   % Interpolate in R
   v0 = 1/(R(J+1)-R(J)) * ( iq_lin(J,K)*(R(J+1)-r) + iq_lin(J+1,K)*(r-R(J)) );
   v1 = 1/(R(J+1)-R(J)) * ( iq_lin(J,K+1)*(R(J+1)-r) + iq_lin(J+1,K+1)*(r-R(J)) );

   % Interpolate in THETA
   val(j,k) = 1/(A(K+1)-A(K)) * ( v0*(A(K+1)-a) + v1*(a-A(K)) );

   end

   end

   end


   % Plot the interpolated values
   figure(1)
   imagesc(abs(val));

   % Take the 2d inverse DFT to get the image
   img = ifft2(val);

   % Plot the image
   figure(2)
   imagesc(abs(img));
```

7.7.2 References

D. C. Munson Jr., J. D. O'Brien, W. K. Jenkins, "A Tomographic Formulation of Spotlight-Mode Synthetic Aperture Radar," Proc. IEEE, vol. 71, pp 917-925, August 1983.

D. C. Munson Jr., R. L. Visentin, "A Signal Processing View of Strip-Mapping Synthetic Aperture Radar," IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 37, no. 12, pp 2131-2147, December 1989.

G. D. Martin, A. W. Doerry, "SAR Polar Format Implementation with MATLAB," Sandia National Laboratories, SAND2005-7413, November 2005.

P. Buxa, L. Gorham, Lt. M. Lukacs, "Mapping of a 2D SAR Backprojection Algorithm to an SRC Reconfigurable Computing MAP Processor," Air Force Research Laboratory, Sensors Directorate.

7.8 THE TEAM!![8]

Our Elec 301 group project team: THE LATE NIGHT ELECS

Aaron Hallquist: ahallquist@rice.edu
Tianlai Lu: tian@rice.edu
Max Magee: max.r.magee@rice.edu
Jason Ryan: jdr@rice.edu

Our pictures can be found on the right!

[8] This content is available online at <http://cnx.org/content/m15668/1.1/>.

Chapter 8

Laugh Track Suppression

8.1 Introduction[1]

This series of modules describes the design, implementation, and analysis of a real-time laugh track removal system. We first give an analysis of the characteristics of a laugh track in both the frequency and time domain, as can be found in the module Anatomy of a Laugh Track (Section 8.2). We then discuss methods for detecting laugh tracks. The primary method can be found in the module Primary Detection Methods for Laugh Tracks (Section 8.3). Another method with which we experimented can be found in the module Secondary Detection Methods for Laugh Tracks (Section 8.4). We conclude with a discussion of the implementation of the real-time component. This can be found in the module DirectShow Filter Design for Laugh Track Removal (Section 8.5). A summary can be found in the Conclusion (Section 8.6)

8.2 Anatomy of a Laugh Track[2]

8.2.1 Introduction

A laugh track is a commonly heard part of sitcoms and other comedy shows found on television since the early 1950s. A laugh track can be defined as a prerecorded superposition of samples of people laughing mixed into a television show to "enhance" it. A single laugh track can be composed of a variety of male laughter and female laughter. Often individual, distinct laughs are added in at the end to make laugh tracks sound like the laughter of a live audience instead of simply something generated from prerecorded samples. All of this makes it very difficult to detect a laugh track based on relatively simple heuristics.

8.2.2 Frequency Analysis

Looking at the Discrete Fourier Transforms of a laugh track(Fig. 1) and a two minute sample of a television show(Fig. 2), we notice some characteristics of the laugh track such as the slightly larger spike in one of the mid-range frequencies, which is difficult to detect, there is not much difference in the spectrum in the two signals. The spikes we see are also not characteristic of every laugh track, so it would be difficult to create a method of detecting laugh tracks solely by looking in the frequency domain. Factoring in variability such as the dominance of male or female voices in the individual laugh track as well as varying lengths and intensities of a laugh, the problem becomes even more difficult when attempting detection using DFTs and a bandpass filtering scheme.

[1] This content is available online at <http://cnx.org/content/m15635/1.5/>.
[2] This content is available online at <http://cnx.org/content/m15644/1.3/>.

TV Show Sample

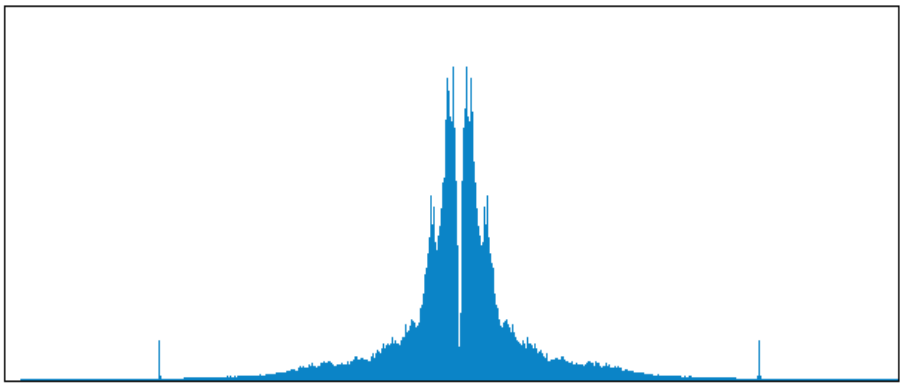


Figure 8.1

Laugh Track Sample

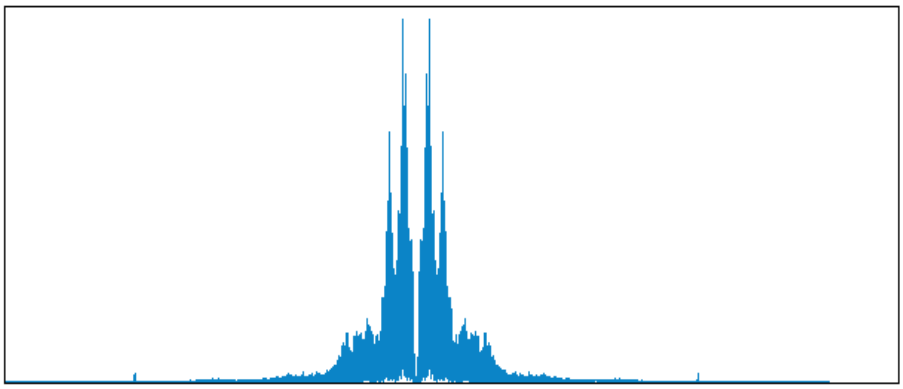


Figure 8.2

8.2.3 Time Analysis

It is much easier to look for a laugh track in the time domain because the envelope that is characteristic of a laugh track as can be seen in the figure below is much more prominent than the spikes that we see in the

DFT of the two signals. This envelope follows the magnitude of a normal laugh track. After a joke, people abruptly start laughing and then the laughter slowly dies down. This is found to be the case in almost every single laugh track instance found in the TV shows that we looked at. Even if a laugh track is short, we can see the same envelope, just compressed more so than in the example. If a laugh track is quieter or louder, we will still be able to see the envelope, though this makes detection based on height thresholds more difficult as regular speech may look very similar to our envelope when a laugh track is shorter and quieter.

Laugh Track in the Time Domain

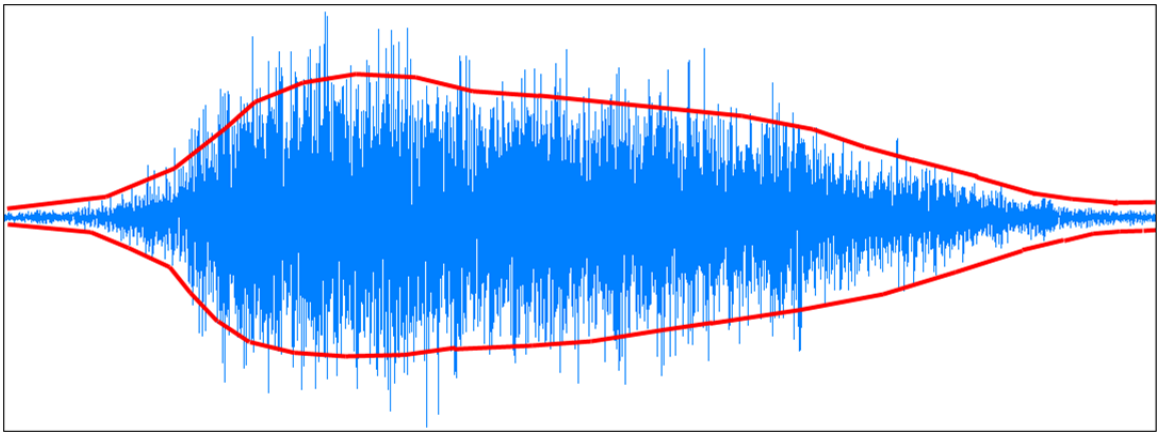


Figure 8.3: A laugh track in the time domain, with the characteristic envelope drawn in red on the original waveform.

8.2.4 Conclusion

Other approaches at detecting laugh tracks were also considered, but envelope detection in the time domain proved to be the most effective manner of detecting laugh tracks. Matched filtering and looking for distinctive characteristics in the frequency domain proved to be fruitless. The envelope in the figure above is quite distinctive and relatively easy to detect, even with a fairly simple algorithm, so this approach was used in our detection and removal scheme.

8.3 Primary Detection Methods for Laugh Tracks[3]

8.3.1 Introduction

Based on the distinct time-domain characteristics of a laugh track, it is possible to use a simple method to detect such sounds. Our primary detection method uses the envelope of the input signal to find laughs.

[3]This content is available online at <http://cnx.org/content/m15642/1.2/>.

8.3.2 Finding the Envelope

8.3.2.1 Method 1

We used two methods of finding and smoothing the envelope of the input signal. In the first method, the magnitude of the input signal is fed into a low pass filter and then squared to obtain the envelope. The filter is a 1000-point fir, linear phase filter generated by MATLAB. The method is simple and easy to implement, but not very efficient.

8.3.2.2 Method 2

Another method of finding the envelope of a signal is by using the Hilbert Transform. The Hilbert Transform shifts all the positive frequencies in a signal forward by pi/2 and all the negative frequencies backward by the same amount. The envelope may then be calculated by taking the square root of the sum of the squares of the Hilbert transform and the original signal. The Hilbert Transform is calculated by taking the FFT of the input signal, multiplying the positive frequencies by j and the negative frequencies by –j, and taking the inverse FFT. As in the first method, the envelope needs to be smoothed for further processing by low-pass filtering.

8.3.3 Locating Laughs

Once the envelope of the signal is found, laughs are detected by a threshold system. The location routine iterates through the samples of the envelope looking for values above a given amplitude threshold. Once this threshold is reached, the routine continues, tracking how long the envelope stays above a second amplitude threshold (lower than the first). If this width reaches a given threshold, the part of the signal from where its envelope rises above the first amplitude threshold to the part where its envelope drops below the second amplitude threshold is flagged as a laugh.

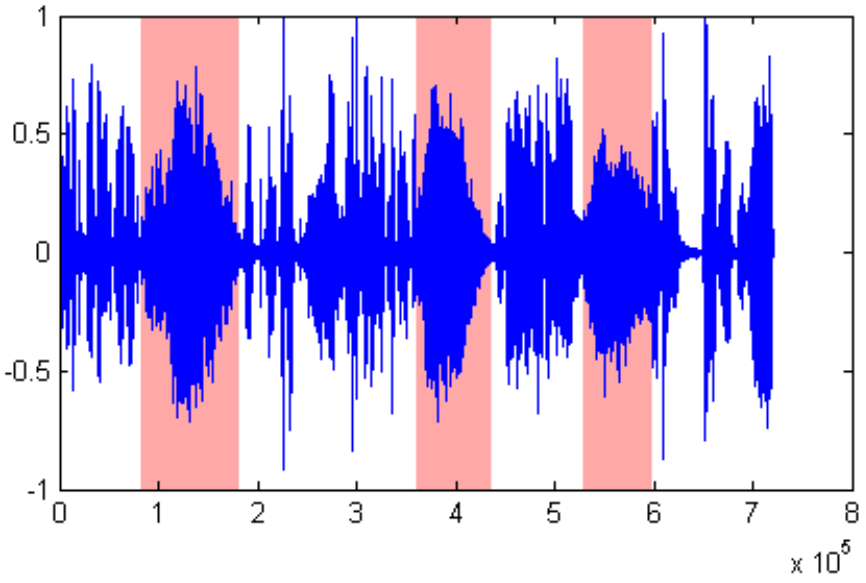


Figure 8.4: Original signal with laughs highlighted

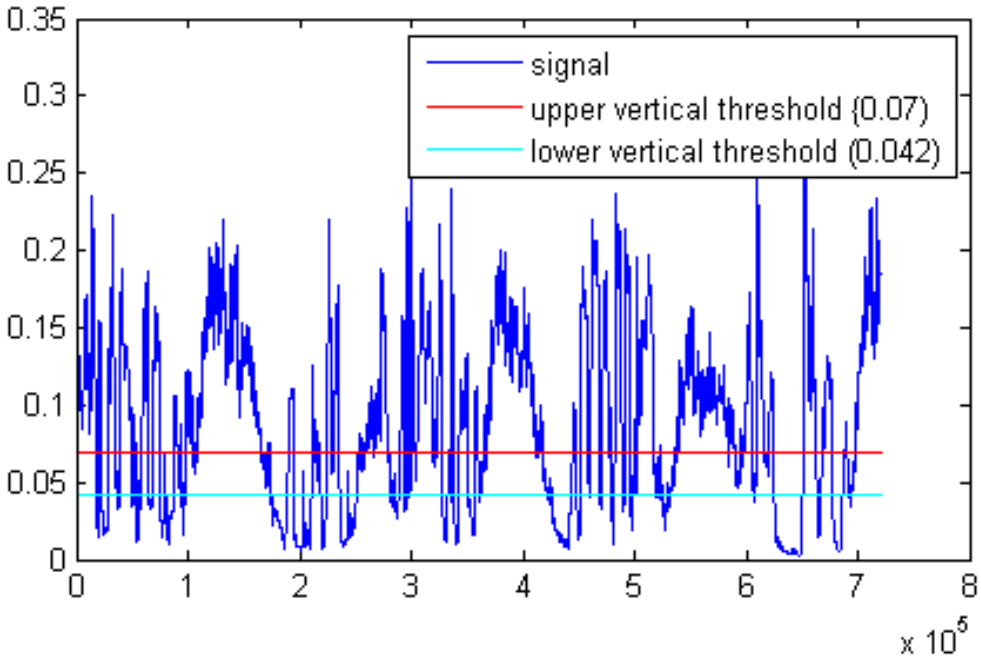


Figure 8.5: Envelope of signal with amplitude thresholds

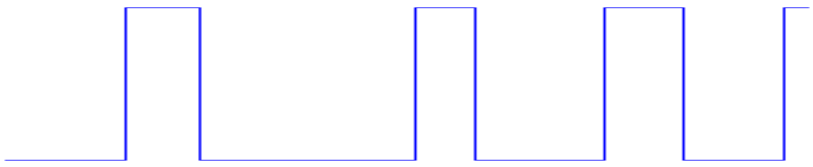


Figure 8.6: Regions of input signal flagged as laughs

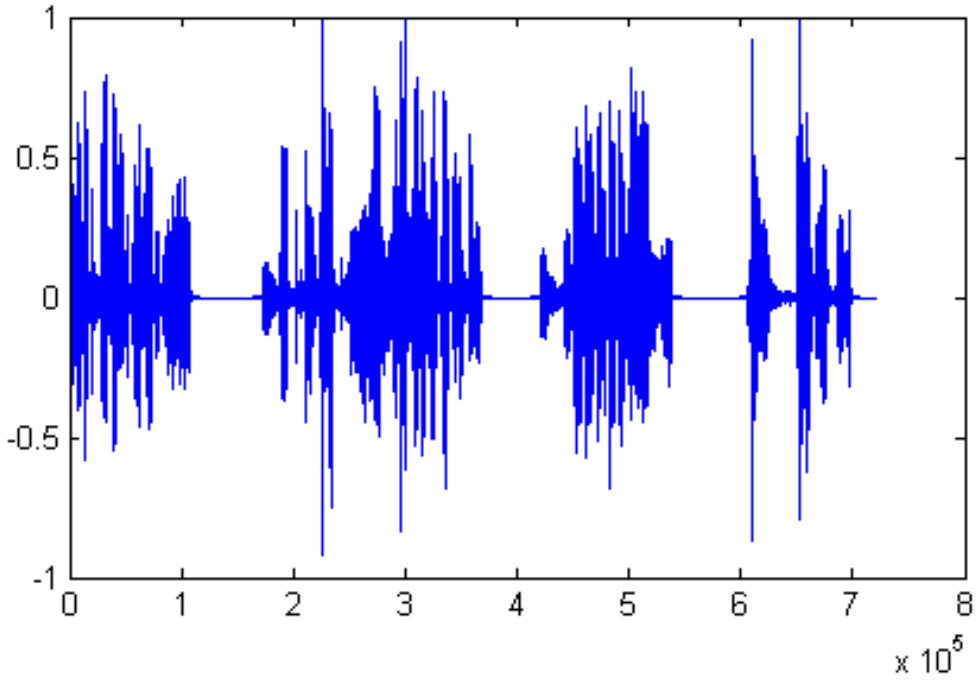


Figure 8.7: Signal with laughs removed

8.3.4 Results

Our primary detection method gave us good results for the sound clips we used (sitcoms with audio tracks consisting mostly of dialogue and laughs). It does however have a few shortcomings. This method has trouble detecting laughs of low volume, and tends to cut out other sounds that overlap with the laughs. Additionally, it has trouble finding precisely the beginning and end of laughs, due to variations in the envelope shapes of different laughs. We had very little trouble with false positives in the clips we tested, however more sophisticated methods would be required to distinguish between laughter and other sounds with similar time-domain characteristics (such as applause).

8.4 Secondary Detection Methods for Laugh Tracks[4]

8.4.1 Introduction

In this module, some of the more exotic detection methods considered for our laugh track removal system are discussed. In particular, a detection method using polynomial curve fitting in conjunction with support vector machines.

[4]This content is available online at <http://cnx.org/content/m15637/1.2/>.

8.4.2 Support Vector Machine (SVM)/Polynomial Curve Fitting

As previously discussed in the Anatomy of a Laugh Track (Section 8.2) module, laugh tracks have a characteristic shape in the time domain. In order to take advantage of this fact, one detection method we considered involved using support vector machines and polynomial curve fitting to detect laughs. Given the unique shape of a laugh track, one ought to be able to consistently fit the same polynomial curve onto a signal representing a laugh. This curve can then be completely characterized in terms of its coefficients. By then building a database of positive and negative examples of polynomial coefficients, and training a support vector machine on this data, we ought to then be able to produce a fast detection scheme for laughs.

8.4.3 What are support vector machines?

Support vector machines are a relatively new technique for partitioning high dimensional datasets into classes. At the simplest level, a SVM divides a high dimensional dataset using hyper-planes. These divisions can then be completely characterized in terms of the data points – called support vectors – that are closest to the hyper-plane of the division. In this way an enormously complex dataset can be partitioned, and more importantly, these partitions are easy to describe. Once a training dataset is partitioned, new points can be classified by checking into which partition the point falls. Further complexity can be introduced by switching from hyper-planes to other curves for partitioning a dataset. Common curves include polynomials, sigmoids, and radial basis functions. More information about support vector machines can be found at www.kernel-machines.org[5] .

8.4.4 Polynomial Curve Fitting

Not to be confused with the polynomial curve potentially used by a SVM, we experimented using polynomial curve fitting to characterize the shape of a laugh track. After experimenting with a handful of potential curves, we chose the degree 9 polynomial as the best curve to use for fitting based on the low error, quick fitting, and high dimension. The high dimension is important, because the power of SVMs becomes most readily apparent in high dimensional space. In this case, our data consisted of 11 dimensional vectors: 9 dimensions from the coefficients of a fitted polynomial plus the duration of the audio segment in question, and the error of the fitted curve.

8.4.5 Implementation

In order to detect laugh tracks, we first divided the audio stream into smaller segments which we then approximated with a polynomial curve and stored into a database for use in a SVM training regime. Observation revealed that the shortest laugh track is approximately 1 second long. To generate our dataset, we considered audio segments spaced slightly apart (0.25 to 0.33 seconds) of at least 1 second in duration. Having fit a curve to the 1 second segment, we then examined a segment slightly longer. If the error of approximation of the longer segment was better, then we considered an even long segment, until an optimal segment was found. The coefficients of the polynomial which was fit, the duration, the error, and the classification (laugh, not laugh), were then stored into a dataset.

Having built a sufficiently large training dataset, we then trained a support vector machine (*LIBSVM*[~cjlin/libsvm/**?**]). Experimentation with parameters showed that the radial basis kernel produced the best results. New audio segments were then classified according to the partitions generated.

8.4.6 Results and Analysis

Overall, the results of this detection method were reasonable, although not ideal. In particular, a 70% detection rate was achieved. Further analysis revealed several factors limited the effectiveness of this method. First, the large number of negative to positive examples heavily skewed the dataset. Experiments showed that

[5] http://www.kernel-machines.org/

results are exceedingly sensitive to parameter selection due to this fact. Second, a large amount of human error existed in the dataset. The training dataset had to be constructed by hand, and variability as to what did and did not qualify as a laugh introduced error. Finally, it is questionable as to how suitable polynomial coefficients are for classifying shape. Coefficient values do not necessarily encode shape characteristics such as duration, slope, and amplitude directly, and as such classification along these lines may be difficult. In sum, this detection method was reasonable, but produced results that were worse than other, simpler methods.

8.5 DirectShow Filter Design for Laugh Track Removal[6]

8.5.1 Real Time Implementation for Laugh Track Removal

8.5.1.1 Overview

In order to make best use of the **Laugh Track Assassinator**'s algorithm, we need to be able to run it in real time with as wide a range of source materials as possible. To accomplish this lofty goal, we have implemented a **DirectShow** filter. DirectShow is Microsoft's technology for manipulating media on the Windows platform. Nearly all media players, such as Windows Media Player, Media Player Classic, and various DVD program, use DirectShow to render video and audio. By writing a DirectShow filter, our algorithm can be used to manipulate nearly any type of media, be it a DVD, an encoded movie, or a live TV video stream.

8.5.1.2 Direct Show

All DirectShow operations are based on filters. Filters describe the translation of data from one source or type to another. DirectShow automatically finds what filters are needed to play a particular media file. The generated graph can be visualized in Microsoft's **GraphEdit** program. Here is what the generated graph looks like for a source video file with the **Laugh Track Assassinator** filter inserted:

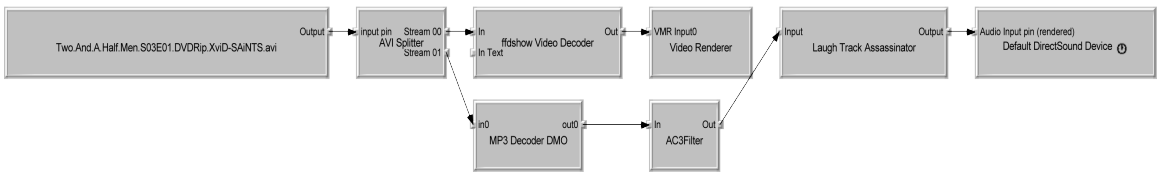


Figure 8.8: This is the filter graph generated by Microsoft DirectShow with the Laugh Track Assassinator filter already inserted.

DirectShow has generated an AVI splitter to transform the file data into an audio and video stream. The video is then sent to the **ffdshow Video Decoder** filter, which is then sent to the **Video Renderer**. The audio stream is sent from the file, through the **MP3 Decoder**, an **AC3Filter**, the **Laugh Track Assassinator**, and finally rendered to the speakers through the **DirectSound** filter.

To create the DirectShow-compatible filter we used Microsoft's **Windows SDK**, and rewrote the audio transform filter example. (The Windows SDK can be downloaded from Microsoft here[7]). We then coded the two main steps in our algorithm: a low pass filter and a threshold detection scheme.

[6] This content is available online at <http://cnx.org/content/m15641/1.5/>.
[7] http://www.microsoft.com/downloads/details.aspx?familyid=4377F86D-C913-4B5C-B87E-EF72E5B4E065&displaylang=en

8.5.1.3 Low Pass Filter

In order to find a balance between frequency resolution and speed, we chose a 1000-point finite impulse response **low pass filter**. We had **Matlab** generate the one thousand filter weights, and then we converted them into a C++ format suitable for DirectShow. Since the filter requires 1000 previous samples to calculate one low pass filtered sample, we created a 1000 point circular buffer to hold the last 1000 samples of the input at any given time.

8.5.1.4 Finite State Machine

The final step in our removal algorithm requires a threshold detection in both amplitude (vertical) and time (horizontal). The requirement for a time-based threshold meant we had to delay the input signal by at least the width of the horizontal threshold. In the end we decided on a 1 second delay to allow for the width threshold of 0.8 seconds, as well as making it easier to resynchronize the video signal with the audio afterward.

The actual threshold test are performed by means of a finite state machine. Here is an overview of the **FSM**:

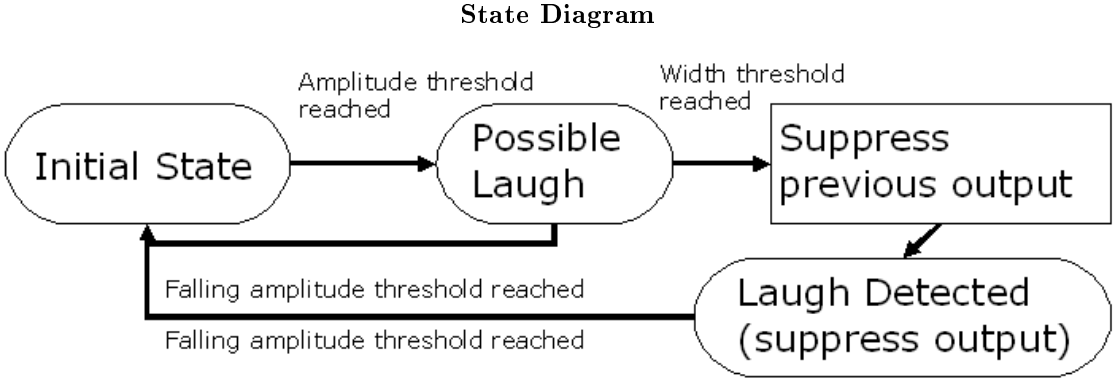


Figure 8.9: This is the state diagram for the real time Laugh Track Assassinator filter.

As soon as the amplitude threshold for the low-passed signal is met, the filter enters the **Possible Laugh** state. From here, if the signal falls below the falling amplitude threshold, the machine returns to the **Initial State**. If the width threshold is reached, then the machine enters the **Laugh Detected** state, and continually suppress the output audio. During this transition, the last second of audio is also eliminated from the output buffer. Since the filter is delayed by at least 1 second, as long as the width threshold is less than this value, the output will reflect the proper changes. Finally, as soon as the falling amplitude threshold is passed, the machine again returns to its **Initial State**.

8.5.1.5 Optimization

The scheme described above generates a working laugh track removal filter. One big problem, however, is speed. Though the above system works on a high-end computer for a real time video signal, any moderate computer will not be able to run it. The chief problem is in the low pass filtering phase.

The low pass filter takes 1000 samples to calculate 1 sample of the low passed signal. This means there are roughly 2000 operations (1000 additions and 1000 multiplications) per sample. With a standard sampling

rate of 44.1 kHz, that means the filter uses 44.1 million operations per second. This is generally unacceptable when accounting for the overhead in the filtering process.

To speed the filter up, we must first realize that we do not need an accurate low pass signal value for every sample. In fact, if we took every 1000 samples of the low pass signal, we would only need to perform 2 operations per sample to get the same results. Using this method gives us a speed increase of 1000x by effectively sampling the low pass filter output. Generally, strictly sampling a signal like this produces rather severe aliasing. But, since the signal is already low-pass-filtered, the signal has already gone anti-aliasing processing, and the optimization works out.

8.5.1.6 Download and Installation

The **Laugh Track Assassinator** filter source code can be downloaded here[8]. The Hamming filter we used can be downloaded here[9]. This is not a complete source listing, but rather the function that handles the actual laughtrack filtering. This code is suitable to be ported to any system that operates on PCM audio streams.

The **Laugh Track Assassinator** filter can be downloaded here[10]. Since this is implemented as a DirectShow filter, this will only run on Windows-based computers.

To install, follow these steps:

- Copy the LaughTrackAssassinator.dll file into your C:\Windows\System32 folder.
- Open a command prompt window (Start->Run->"cmd").
- Type "regsvr32 LaughTrackAssassinator.dll" and press enter in the command box.
- The Laugh Track Assassinator is now registered with DirectShow.

Now that the filter is registered, most any DirectShow based media player should be able to use the filter on any media. We tested the filter with **Media Player Classic**, a free media player that can be downloaded here[11] . Here are the steps to get it to work:

- Open Media Player Classic.
- Go to View->Options->External Filters.
- Select "Add filter...".
- Select the **Laugh Track Assassinator** from the list of available filters.
- Select the newly added filter, and select the "Prefer" radio button.

You can now view any media that has audio and it will automatically run the **Laugh Track Assassinator**. In order to get video back in sync with the audio, you can set the audio delay to 500ms in Media Player Classic by using the + and − keys on the numpad of your keyboard.

8.6 Conclusion[12]

[8] See the file at <http://cnx.org/content/m15641/latest/redist.cpp>
[9] See the file at <http://cnx.org/content/m15641/latest/lpfilter.cpp>
[10] See the file at <http://cnx.org/content/m15641/latest/LaughTrackAssassinator.dll>
[11] http://sourceforge.net/project/showfiles.php?group_id=82303&package_id=84358
[12] This content is available online at <http://cnx.org/content/m15643/1.2/>.

8.6.1 Summary

We have looked at the various ways that a laugh track can be detected and how to implement a laugh track detection algorithm as a Directshow filter. The most effective way of detecting a laugh track appears to be looking at the waveform in the time domain instead of in the frequency domain. The algorithm to use for envelope detection can be changed and become increasingly more sophisticated as we attempt to catch and process correctly some of the more difficult cases.

8.6.2 Detection Techniques

There are multiple ways to generate the envelope we use for detecting a laugh track. The simplest method is to square the signal and then low pass filter it, which gives quite good results. An even better algorithm involves using a Hilbert transform to generate the envelope, but this was too computationally expensive to implement in our Directshow filter.

We have also briefly discussed some of the ways to detect the envelope once we have generated a signal to pick out laughs from. On the simpler range of the spectrum, we can have a width and height thresholds for the laugh portion of the signal's envelope. On the other hand, we could also fit laugh tracks to polynomial curves and then use a Support Vector Machine to detect laugh tracks based on a database of positive and negative matches for laugh tracks.

In the end, we found that detecting and removing laugh tracks from an audio signal is much more complicated than it may appear on the surface. As a human, it is very easy to spot a laugh in the signal, but to create a system that can do this automatically is more complicated. Another interesting aspect of the system is how to remove the laugh tracks, which can be complicated because people often talk over the laugh tracks in our signals. To remove only the laugh track while leaving in the human voice signals is something that needs further exploration.

Index of Keywords and Terms

Keywords are listed by the section with that keyword (page numbers are in parentheses). Keywords do not necessarily appear in the text of the page. They are merely associated with that section. *Ex.* apples, § 1.1 (1) **Terms** are referenced by the page they appear on. *Ex.* apples, 1

Attributions

Collection: *ELEC 301 Projects Fall 2007*
Edited by: Rice University ELEC 301
URL: http://cnx.org/content/col10503/1.1/
License: http://creativecommons.org/licenses/by/2.0/

Module: "Introduction"
By: Thomas Yeh
URL: http://cnx.org/content/m15686/1.1/
Page: 1
Copyright: Thomas Yeh
License: http://creativecommons.org/licenses/by/2.0/

Module: "Image Acquisition"
By: Thomas Yeh
URL: http://cnx.org/content/m15687/1.1/
Pages: 1-5
Copyright: Thomas Yeh
License: http://creativecommons.org/licenses/by/2.0/

Module: "Defining Borders"
By: Thomas Yeh
URL: http://cnx.org/content/m15688/1.1/
Pages: 5-6
Copyright: Thomas Yeh
License: http://creativecommons.org/licenses/by/2.0/

Module: "Filtering"
By: Thomas Yeh
URL: http://cnx.org/content/m15689/1.1/
Pages: 7-8
Copyright: Thomas Yeh
License: http://creativecommons.org/licenses/by/2.0/

Module: "Connecting Lines"
By: Thomas Yeh
URL: http://cnx.org/content/m15690/1.1/
Pages: 8-9
Copyright: Thomas Yeh
License: http://creativecommons.org/licenses/by/2.0/

Module: "Edge Detection"
By: Thomas Yeh
URL: http://cnx.org/content/m15691/1.1/
Pages: 9-10
Copyright: Thomas Yeh
License: http://creativecommons.org/licenses/by/2.0/

Module: "Hough Transform"
By: Thomas Yeh
URL: http://cnx.org/content/m15692/1.1/
Pages: 10-12
Copyright: Thomas Yeh
License: http://creativecommons.org/licenses/by/2.0/

Module: "Conclusion"
By: Thomas Yeh
URL: http://cnx.org/content/m15693/1.1/
Pages: 12-13
Copyright: Thomas Yeh
License: http://creativecommons.org/licenses/by/2.0/

Module: "Team"
By: Thomas Yeh
URL: http://cnx.org/content/m15694/1.3/
Pages: 14-17
Copyright: Thomas Yeh
License: http://creativecommons.org/licenses/by/2.0/

Module: "Introduction to the Virtual Drum Kit"
By: Janice Chow, Kriti Charan, Tanwee Misra
URL: http://cnx.org/content/m15698/1.1/
Pages: 19-20
Copyright: Janice Chow, Kriti Charan, Tanwee Misra
License: http://creativecommons.org/licenses/by/2.0/

Module: "A General Approach to Building Your Own Virtual Drum Kit"
By: Janice Chow, Kriti Charan, Tanwee Misra
URL: http://cnx.org/content/m15695/1.1/
Pages: 21-23
Copyright: Janice Chow, Kriti Charan, Tanwee Misra
License: http://creativecommons.org/licenses/by/2.0/

Module: "Webcam Color Tracking in Matlab"
By: Janice Chow, Kriti Charan, Tanwee Misra
URL: http://cnx.org/content/m15696/1.2/
Pages: 23-26
Copyright: Janice Chow, Kriti Charan, Tanwee Misra
License: http://creativecommons.org/licenses/by/2.0/

Module: "Implementation: Detecting a Hit"
By: Janice Chow
URL: http://cnx.org/content/m15714/1.1/
Pages: 26-28
Copyright: Janice Chow
License: http://creativecommons.org/licenses/by/2.0/

Module: "Implementation: Detecting the LEDs"
By: Tanwee Misra, Kriti Charan, Janice Chow
URL: http://cnx.org/content/m15699/1.1/
Pages: 29-31
Copyright: Tanwee Misra, Kriti Charan, Janice Chow
License: http://creativecommons.org/licenses/by/2.0/

Module: "Producing Drum Sounds and Displaying the Drum"
By: Tanwee Misra, Kriti Charan, Janice Chow
URL: http://cnx.org/content/m15697/1.1/
Pages: 31-32
Copyright: Tanwee Misra, Kriti Charan, Janice Chow
License: http://creativecommons.org/licenses/by/2.0/

Module: "Implementation: Code"
By: Tanwee Misra, Kriti Charan, Janice Chow
URL: http://cnx.org/content/m15704/1.1/
Page: 33
Copyright: Tanwee Misra, Kriti Charan, Janice Chow
License: http://creativecommons.org/licenses/by/2.0/

Module: "Results of the Virtual Drum Kit"
By: Kriti Charan, Tanwee Misra, Janice Chow
URL: http://cnx.org/content/m15711/1.1/
Page: 33
Copyright: Kriti Charan, Tanwee Misra, Janice Chow
License: http://creativecommons.org/licenses/by/2.0/

Module: "Future Work for the Virtual Drum Kit"
By: Kriti Charan, Tanwee Misra, Janice Chow
URL: http://cnx.org/content/m15709/1.1/
Page: 34
Copyright: Kriti Charan, Tanwee Misra, Janice Chow
License: http://creativecommons.org/licenses/by/2.0/

Module: "Team Members and Acknowledgments"
By: Kriti Charan, Tanwee Misra, Janice Chow
URL: http://cnx.org/content/m15713/1.1/
Page: 34
Copyright: Kriti Charan, Tanwee Misra, Janice Chow
License: http://creativecommons.org/licenses/by/2.0/

Module: "Blind Source Separation Via ICA: Introduction and Background"
By: Angela Qian, John Steinbauer, Akshay Dayal, Mark Eastaway
URL: http://cnx.org/content/m15702/1.1/
Page: 35
Copyright: Angela Qian, John Steinbauer, Akshay Dayal, Mark Eastaway
License: http://creativecommons.org/licenses/by/2.0/

Module: "Blind Source Separation Via ICA: Implementation"
By: Mark Eastaway, John Steinbauer, Angela Qian, Akshay Dayal
URL: http://cnx.org/content/m15639/1.2/
Pages: 36-37
Copyright: Mark Eastaway, John Steinbauer, Angela Qian, Akshay Dayal
License: http://creativecommons.org/licenses/by/2.0/

Module: "Blind Source Separation Via ICA: Math Behind Method"
By: Akshay Dayal, John Steinbauer, Angela Qian, Mark Eastaway
URL: http://cnx.org/content/m15708/1.1/
Pages: 37-39
Copyright: Akshay Dayal, John Steinbauer, Angela Qian, Mark Eastaway
License: http://creativecommons.org/licenses/by/2.0/

Module: "Blind Source Separation Via ICA: Methods and Trials"
By: John Steinbauer, Mark Eastaway, Akshay Dayal, Angela Qian
URL: http://cnx.org/content/m15638/1.3/
Pages: 39-42
Copyright: John Steinbauer, Mark Eastaway, Akshay Dayal, Angela Qian
License: http://creativecommons.org/licenses/by/2.0/

Module: "Blind Source Separation Via ICA: Audio Demonstration"
By: John Steinbauer, Akshay Dayal, Mark Eastaway, Angela Qian
URL: http://cnx.org/content/m15712/1.1/
Pages: 42-43
Copyright: John Steinbauer, Akshay Dayal, Mark Eastaway, Angela Qian
License: http://creativecommons.org/licenses/by/2.0/

Module: "Blind Source Separation Via ICA: Signal Processing Applications"
By: Mark Eastaway, John Steinbauer, Angela Qian, Akshay Dayal
URL: http://cnx.org/content/m15640/1.3/
Pages: 43-46
Copyright: Mark Eastaway, John Steinbauer, Angela Qian, Akshay Dayal
License: http://creativecommons.org/licenses/by/2.0/

Module: "Blind Source Separation Via ICA: Medical Imaging Applications"
By: Angela Qian, John Steinbauer, Akshay Dayal, Mark Eastaway
URL: http://cnx.org/content/m15701/1.1/
Pages: 46-47
Copyright: Angela Qian, John Steinbauer, Akshay Dayal, Mark Eastaway
License: http://creativecommons.org/licenses/by/2.0/

Module: "Blind Source Separation Via ICA: Conclusion"
By: Akshay Dayal, John Steinbauer, Mark Eastaway, Angela Qian
URL: http://cnx.org/content/m15706/1.1/
Page: 48
Copyright: Akshay Dayal, John Steinbauer, Mark Eastaway, Angela Qian
License: http://creativecommons.org/licenses/by/2.0/

Module: "Blind Source Separation Via ICA: The Team and Thanks"
By: Angela Qian, John Steinbauer, Akshay Dayal, Mark Eastaway
URL: http://cnx.org/content/m15703/1.1/
Pages: 48-49
Copyright: Angela Qian, John Steinbauer, Akshay Dayal, Mark Eastaway
License: http://creativecommons.org/licenses/by/2.0/

Module: "Introduction"
By: Jason Holden
URL: http://cnx.org/content/m15680/1.1/
Page: 51
Copyright: Jason Holden
License: http://creativecommons.org/licenses/by/2.0/

Module: "Setup"
By: Jason Holden
URL: http://cnx.org/content/m15682/1.1/
Pages: 51-52
Copyright: Jason Holden
License: http://creativecommons.org/licenses/by/2.0/

Module: "Implementation"
By: Naren Anand
URL: http://cnx.org/content/m15679/1.1/
Pages: 52-54
Copyright: Naren Anand
License: http://creativecommons.org/licenses/by/2.0/

Module: "Inverse Filter"
By: CJ Steuernagel
URL: http://cnx.org/content/m15685/1.1/
Pages: 54-59
Copyright: CJ Steuernagel
License: http://creativecommons.org/licenses/by/2.0/

Module: "Vocal Band Pass Filter"
By: Jason Holden
URL: http://cnx.org/content/m15683/1.1/
Pages: 59-61
Copyright: Jason Holden
License: http://creativecommons.org/licenses/by/2.0/

Module: "Results"
By: Jason Holden
URL: http://cnx.org/content/m15684/1.1/
Pages: 61-62
Copyright: Jason Holden
License: http://creativecommons.org/licenses/by/2.0/

Module: "MATLAB EQ: Problem"
By: Alex Mrozack
URL: http://cnx.org/content/m15652/1.1/
Page: 63
Copyright: Alex Mrozack
License: http://creativecommons.org/licenses/by/2.0/

Module: "MATLAB EQ: Background on Equalization"
By: Chris Corbet
URL: http://cnx.org/content/m15655/1.1/
Pages: 63-66
Copyright: Chris Corbet, Alex Mrozack
License: http://creativecommons.org/licenses/by/2.0/

Module: "MATLAB EQ: Sound Sampling"
By: Chris Corbet
URL: http://cnx.org/content/m15657/1.1/
Pages: 66-67
Copyright: Chris Corbet, Alex Mrozack
License: http://creativecommons.org/licenses/by/2.0/

Module: "MATLAB EQ: Approach: Use of Frequency Binning to Apply Frequency Gains"
By: Alex Mrozack
URL: http://cnx.org/content/m15650/1.1/
Pages: 67-68
Copyright: Alex Mrozack
License: http://creativecommons.org/licenses/by/2.0/

Module: "MATLAB EQ: Approach: Time-Domain and Effects"
By: Niklos Maureira
URL: http://cnx.org/content/m15654/1.1/
Pages: 68-73
Copyright: Niklos Maureira, Alex Mrozack
License: http://creativecommons.org/licenses/by/2.0/

Module: "MATLAB EQ: Results"
By: Niklos Maureira
URL: http://cnx.org/content/m15656/1.1/
Page: 74
Copyright: Niklos Maureira, Alex Mrozack
License: http://creativecommons.org/licenses/by/2.0/

Module: "MATLAB EQ: The Team"
By: Niklos Maureira, Alex Mrozack
URL: http://cnx.org/content/m15653/1.2/
Pages: 75-76
Copyright: Niklos Maureira, Alex Mrozack
License: http://creativecommons.org/licenses/by/3.0/

Module: "Introduction"
By: Blake Brogdon, Thomas Deitch, Kyle Barnhart, Britt Antley
URL: http://cnx.org/content/m15674/1.2/
Page: 77
Copyright: Blake Brogdon, Thomas Deitch, Kyle Barnhart, Britt Antley
License: http://creativecommons.org/licenses/by/2.0/

Module: "Algorithm Overview"
By: Blake Brogdon, Thomas Deitch, Kyle Barnhart, Britt Antley
URL: http://cnx.org/content/m15673/1.2/
Pages: 77-80
Copyright: Blake Brogdon, Thomas Deitch, Kyle Barnhart, Britt Antley
License: http://creativecommons.org/licenses/by/2.0/

Module: "Least Mean Squares Adaptive Filters"
By: Blake Brogdon, Thomas Deitch, Kyle Barnhart, Britt Antley
URL: http://cnx.org/content/m15675/1.2/
Pages: 80-81
Copyright: Blake Brogdon, Thomas Deitch, Kyle Barnhart, Britt Antley
License: http://creativecommons.org/licenses/by/2.0/

Module: "Joint Time-Frequency Analysis"
By: Blake Brogdon, Thomas Deitch, Kyle Barnhart, Britt Antley
URL: http://cnx.org/content/m15676/1.2/
Pages: 82-85
Copyright: Blake Brogdon, Thomas Deitch, Kyle Barnhart, Britt Antley
License: http://creativecommons.org/licenses/by/2.0/

Module: "iTunes Control Interface"
By: Blake Brogdon, Thomas Deitch, Kyle Barnhart, Britt Antley
URL: http://cnx.org/content/m15677/1.2/
Page: 85
Copyright: Blake Brogdon, Thomas Deitch, Kyle Barnhart, Britt Antley
License: http://creativecommons.org/licenses/by/2.0/

Module: "Conclusion and Possible Improvements"
By: Blake Brogdon, Thomas Deitch, Kyle Barnhart, Britt Antley
URL: http://cnx.org/content/m15678/1.2/
Page: 85
Copyright: Blake Brogdon, Thomas Deitch, Kyle Barnhart, Britt Antley
License: http://creativecommons.org/licenses/by/2.0/

Module: "Introduction"
By: Aaron Hallquist
URL: http://cnx.org/content/m15658/1.1/
Page: 87
Copyright: Aaron Hallquist
License: http://creativecommons.org/licenses/by/2.0/

Module: "Background"
By: Tianlai Lu
URL: http://cnx.org/content/m15667/1.1/
Pages: 87-88
Copyright: Tianlai Lu
License: http://creativecommons.org/licenses/by/2.0/

Module: "Projection-Slice Theorem"
By: Jason Ryan
URL: http://cnx.org/content/m15663/1.1/
Pages: 88-91
Copyright: Jason Ryan
License: http://creativecommons.org/licenses/by/2.0/

Module: "Tomographic Processing"
By: Jason Ryan
URL: http://cnx.org/content/m15661/1.1/
Pages: 91-97
Copyright: Jason Ryan
License: http://creativecommons.org/licenses/by/2.0/

Module: "Data and Processing"
By: Aaron Hallquist
URL: http://cnx.org/content/m15659/1.1/
Pages: 98-100
Copyright: Aaron Hallquist
License: http://creativecommons.org/licenses/by/2.0/

Module: "Conclusions and Future Work"
By: Aaron Hallquist
URL: http://cnx.org/content/m15660/1.1/
Pages: 100-102
Copyright: Aaron Hallquist
License: http://creativecommons.org/licenses/by/2.0/

Module: "Appendix and References"
By: Tianlai Lu
URL: http://cnx.org/content/m15665/1.1/
Pages: 102-103
Copyright: Tianlai Lu
License: http://creativecommons.org/licenses/by/2.0/

Module: "THE TEAM!!"
By: Tianlai Lu
URL: http://cnx.org/content/m15668/1.1/
Page: 104
Copyright: Tianlai Lu
License: http://creativecommons.org/licenses/by/2.0/

Module: "Introduction"
By: Taylor Goodhart
URL: http://cnx.org/content/m15635/1.5/
Page: 105
Copyright: Taylor Goodhart
License: http://creativecommons.org/licenses/by/2.0/

Module: "Anatomy of a Laugh Track"
By: Oleg Pesok
URL: http://cnx.org/content/m15644/1.3/
Pages: 105-107
Copyright: Oleg Pesok
License: http://creativecommons.org/licenses/by/2.0/

Module: "Primary Detection Methods for Laugh Tracks"
By: Keith Wilhelm
URL: http://cnx.org/content/m15642/1.2/
Pages: 107-110
Copyright: Keith Wilhelm
License: http://creativecommons.org/licenses/by/2.0/

Module: "Secondary Detection Methods for Laugh Tracks"
By: Taylor Goodhart
URL: http://cnx.org/content/m15637/1.2/
Pages: 110-112
Copyright: Taylor Goodhart
License: http://creativecommons.org/licenses/by/2.0/

Module: "DirectShow Filter Design for Laugh Track Removal"
By: Justin Nordin
URL: http://cnx.org/content/m15641/1.5/
Pages: 112-114
Copyright: Justin Nordin
License: http://creativecommons.org/licenses/by/2.0/

Module: "Conclusion"
By: Oleg Pesok
URL: http://cnx.org/content/m15643/1.2/
Pages: 114-115
Copyright: Oleg Pesok
License: http://creativecommons.org/licenses/by/2.0/

About Connexions

Since 1999, Connexions has been pioneering a global system where anyone can create course materials and make them fully accessible and easily reusable free of charge. We are a Web-based authoring, teaching and learning environment open to anyone interested in education, including students, teachers, professors and lifelong learners. We connect ideas and facilitate educational communities.

Connexions's modular, interactive courses are in use worldwide by universities, community colleges, K-12 schools, distance learners, and lifelong learners. Connexions materials are in many languages, including English, Spanish, Chinese, Japanese, Italian, Vietnamese, French, Portuguese, and Thai. Connexions is part of an exciting new information distribution system that allows for **Print on Demand Books**. Connexions has partnered with innovative on-demand publisher QOOP to accelerate the delivery of printed course materials and textbooks into classrooms worldwide at lower prices than traditional academic publishers.