Turning Images into Sounds using Inverse Fourier Transforms

Project Design Document

Orlala Wentink and Elana Voigt

12/12/2014

Contents

1. Introduction

a. Purpose

To input an arbitrary image and process it into a sound wave. This is related to processes done in spectroscopy and by radio telescopes. Musicians have also used this idea to make music out of images.

b. Scope

This software aims to extract the pixel data of an arbitrary image and to convert that image into sound through Inverse Fast Fourier Transforms. The output is a DAQ speaker that processes a voltage measurement and outputs sound.

c. Definitions and Acronyms

LV: LabView 2014 32-bit
FT: Fourier Transform
FFT: Fast Fourier Transform
IFFT: Inverse Fast Fourier Transform
DAQ: The DAQ used in this program is a National Instruments USB 6341
RGB: Red/Green/Blue, refers to color data produced by the reading of a JPEG file.
STFT: Short Time Fourier Transform
ISTFT: Inverse Short Time Fourier Transform
GASE: Gamma ray burst All Sky Experiment

2. System Overview

The theoretical basis for transforming an image into sound is rooted in FT. We used the principles of STFT, a technique particularly used in spectroscopy. Spectroscopy derives a spectrogram from a measured wave signal (electromagnetic, sound). The spectrogram below (Figure 1) is real spectrogram data from GASE, supervised by Prof Morales, located on Seig Hall. This spectrogram is of approximately the 0-60 MHz range ("chan", short for channel, 0 corresponds to 0 MHz, and channel 1000 corresponds to approximately 60 MHz, a single channel bins approximately .06 MHz). The time scale, labeled in integrations, is approximately a 3.5 day range (1 time integration is 10 seconds, so it is essentially binning time as well). The color scale corresponds to the power (measured by the antennas as voltage). This is produced by the antennas naturally performing a FT on the sky. Fourier Transformation is entirely reversable, and this is the basis of our project. Instead of creating a spectrogram from sound or electromagnetic radiation we are reading images as spectrograms and transforming them into sound. Inversing a STFT you pick a dt to look at and you do a IFFT on the frequency information contained in that time slice, transforming it to amplitude of the constituent frequency vs time. An example of what this time slice looks like is in Figure 3, graphed as power vs. frequency. In our implementation this would correspond to a column or columns of pixel data.

References

"Short-time Fourier transform". Wikipedia. Accessed 12/12/2014. http://en.wikipedia.org/wiki/Short-time_Fourier_transform

Please note: I do not normally reference wikipedia, however this is where Prof Burnett and our TA Durmus Karatay sent us.

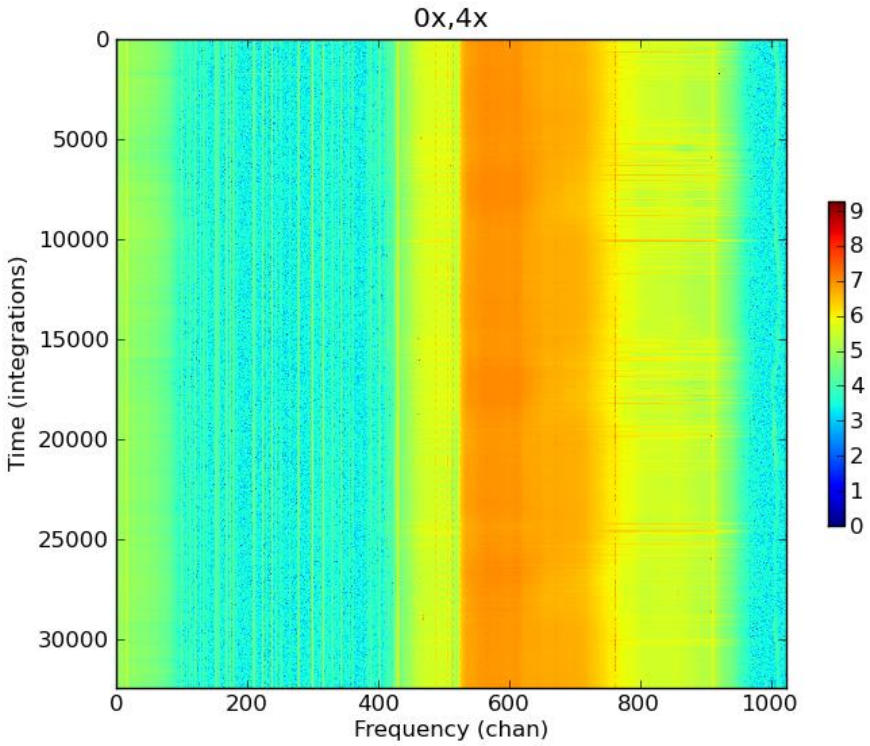


Figure 1: Spectrogram data from the GRB All Sky Experiment (GASE), supervised by Prof Morales, located on Seig Hall, taken in summer of 2014.

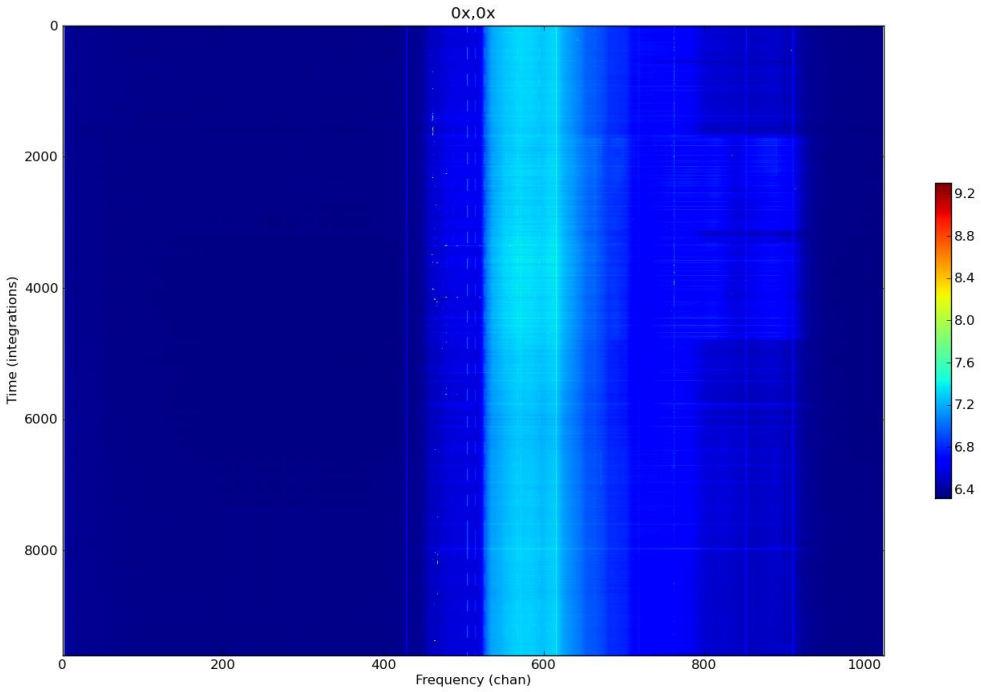


Figure 2: Spectrogram data from the GRB All Sky Experiment (GASE), supervised by Prof Morales, located on Seig Hall, taken in summer of 2014 after the data in figure 1.

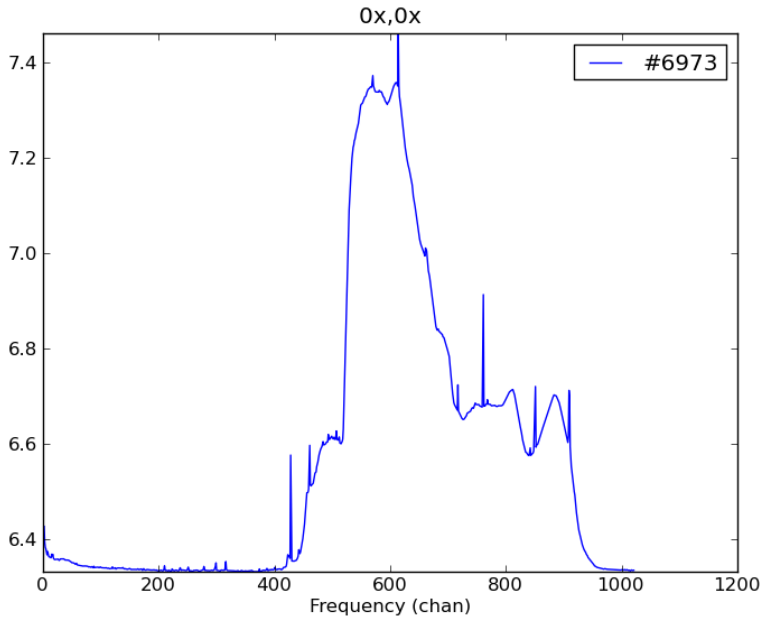


Figure 3: Time slice of a time integration #6973 from Figure 2. This is representative of what a dt slice pulled out to be run through IFFT would look like if done on the spectrogram in Figure 2. The power scale is logarithmic.

3. System Architecture

a. Architectural Design

There is a primary VI “ImageToSound” that is the only VI the user has to interact with. ImageToSound passes a selected JPEG file to a sub-VI, “ImageInputToMatrix”, which processes the image and returns an array of the pixel color data and the image dimension properties. ImageToSound outputs an audio signal to either the DAQ or computer speakers, displays a graph of the audio waveform, and displays the selected image.

ImageInputToMatrix puts the image through a JPEG file reader which produces the dimensions of the image and a 1-D array of color values. The array is in RGB format, meaning for every pixel of the image has 3 entries in the image data array, one for each color. Through a loop structure the absolute color intensity for each pixel is determined, using a standard gray-scale conversion formula:

intensity in gray = 0.2989 * intensity red + 0.5870 * intensity green + 0.1140 * intensity blue

Then the 1-D array of the pixel color intensity is passed back to the ImageToSound VI. The ImageToSound VI then puts the array of gray-scaled image data into a for-loop. The loop runs as many times as the number of columns of pixels in the image, this value is labeled as “right” in the image data.

Each loop extracts the number of array entries corresponding to the length of one column of pixels in the image, this value is labeled as “bottom” in the image data. An IFFT of the selected data is then performed in the loop. This corresponds to the first step in performing ISTFT. The first data point of every FT-ed data set is discarded. We chose to discard it because it gave us an unreasonable value, consistently. This is expected since FT is best with continuous data, so there are always boundary artifacts in practical FT of real data. We then normalize the FT-ed data to a -10 to 10 scale. We did this by finding the largest absolute value in the array and dividing the entire array by that value, then multiplying the array by 10. This array is then passed to Build Waveform which also requires a dt input. The dt is controled via the inverse of the sampling rate, which is defined by the user. Build Waveform uses dt to actually do the frequency assignments for the IFFT values. The entire IFFT and Build Waveform process is actually performing an ISTFT. Build Waveform outputs waveform data to a graph, and a case structure selection which allows the user to pick sound output either to the DAQ or computer speakers.

To set up the DAQ, wire pin 15 as V+, pin 16 and V0, and run those to a BNC cable on a breadboard. Then run that cable to speaker in and wire a speaker to speaker out on the speaker set up from the Kundt’s tube experiment. You may have to hold the speaker up to your ear, the output is somewhat quiet.

b. Design Rationale

We chose the 1-D array output for the ImageInputToMatrix because there were severe implementation problems with trying to read the data as a matrix or 2-D array. They could not be passed into the IFFT and then passed easily to waveform input, it had too many dimensions in the IFFT output. In fact, the matrix form could not be passed into the IFFT we used at all. Although the matrix format is easier to display and understand as a representation of a picture, it was very hard to process. The 1-D array is confusing to understand at first as a picture

representation, but easier to work with in the program architecture. There is some processing in ImageInputToMatrix that converts the array to a matrix and a 2-D array, however this is not passed back or used at this time.

The normalization structure was chosen after using a variety of pictures and determining that we had to ensure normalization was done accounting for the largest absolute value in the array, not just the largest positive value. The case structure was chosen because it seemed the easiest way to have a user blind selection of the largest absolute value for normalization. Normalization itself was made necessary by the limitations of the DAQ. While the array will accept a large range of values, the DAQ input requires voltage values limited to between -10 and 10, so the amplitudes had to be normalized to that range.

The choice to have the user input a sampling frequency was based on our attempt to recreate a known sentence from a waveform used by linguists. Initially we thought this would allow us to test our code to see if it was providing a "true" representation of the image in sound form. However, we discovered that without knowing the sampling rate the waveform was created at we could not successfully recreate it. The sampling frequency also determines how long the transformed image will play for as sound, so we decided to leave it in as a user control. When reading an arbitrary image it allows the user to define how long the sound plays. A lower sampling frequency corresponds to a longer audio file. If we created a complementary VI to make an image out of sound, we could then determine how well our process is directly processing the image. We would also then use the dt from the audio to image STFT to put into the Build Waveform in ImageToSound.

A big component of our work relied on trouble shooting using Matlab. For example, a JPEG image is encoded with data for red, green, and blue channels – each pixel gets three numbers, one for each color, where each number is from 0 to 256. Matlab displays this information in a 3-D matrix, where each 2-D matrix holds the information for each individual color channel. You can take each matrix and display it as an image. The images on the pages that follow are the original image, matrix 1, matrix 2, matrix 3, and just for reference the grayscale version of the image using the Matlab command '**rbg2gray**'. We assume that matrices 1, 2, and 3 correspond to red, green, and blue, respectively. Note that when displaying a single matrix at a time, it displays in grayscale, and each image is slightly different – the easiest place to see this is in the shade of the sky.

This knowledge was extremely valuable to us in understanding the way that LV displays the image data; instead of displaying the data in a 3-D matrix, LV displays it in a 1-D array. Looking at the matrices in Matlab allowed us to figure out in what order LV was displaying the data. It turns out that the data is stored as [R11, G11, B11, R12, G12, B12…] as opposed to all of the red data first, then the green data, then the blue (where, for example, R12 is Red, row 1, column 2).

Also, looking at the matrix dimensions helped us determine how "right" and "bottom" in the image data corresponded to columns and rows in the matrix representation. This allowed to make sense of how the array output corresponded to the image data, an essential component of our loop structure design.

Original Image

Matrix 1 (Red)

Matrix 2 (Green)

Matrix 3 (Blue)

Grayscale: using Matlab function '**rgb2gray**'

Our submitted files also include 3 diagnostic VI's. ImageToSoundWithDiagnostics has the exact same functionality as ImageToSound, but includes arrays and array sizes, as well as graphs for many points in the data stream to allow for diagnostics. The other 2 VI's were the start of our attempt to determine how to process sound into an image and back to sound, and particularly to explore the issues around determining dt. As it stands, all it does is FFT a WAV file, then IFFT that same file and play it through the DAQ.

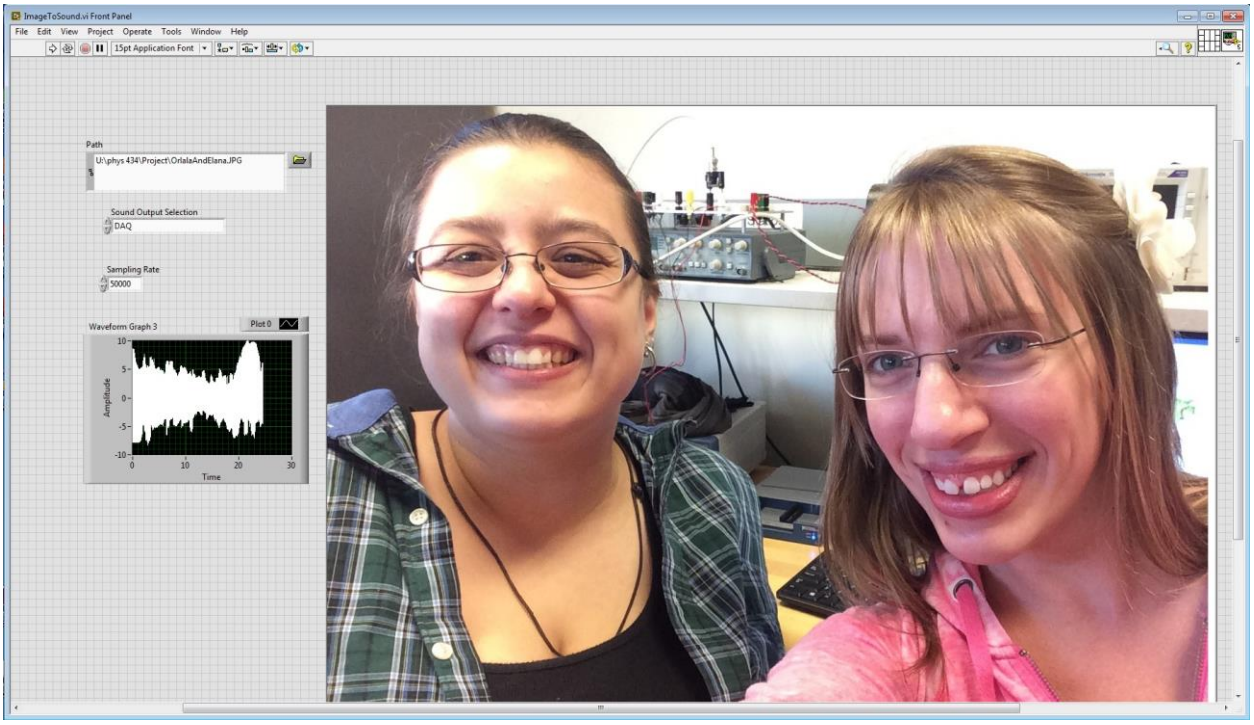
4. Human Interface Design

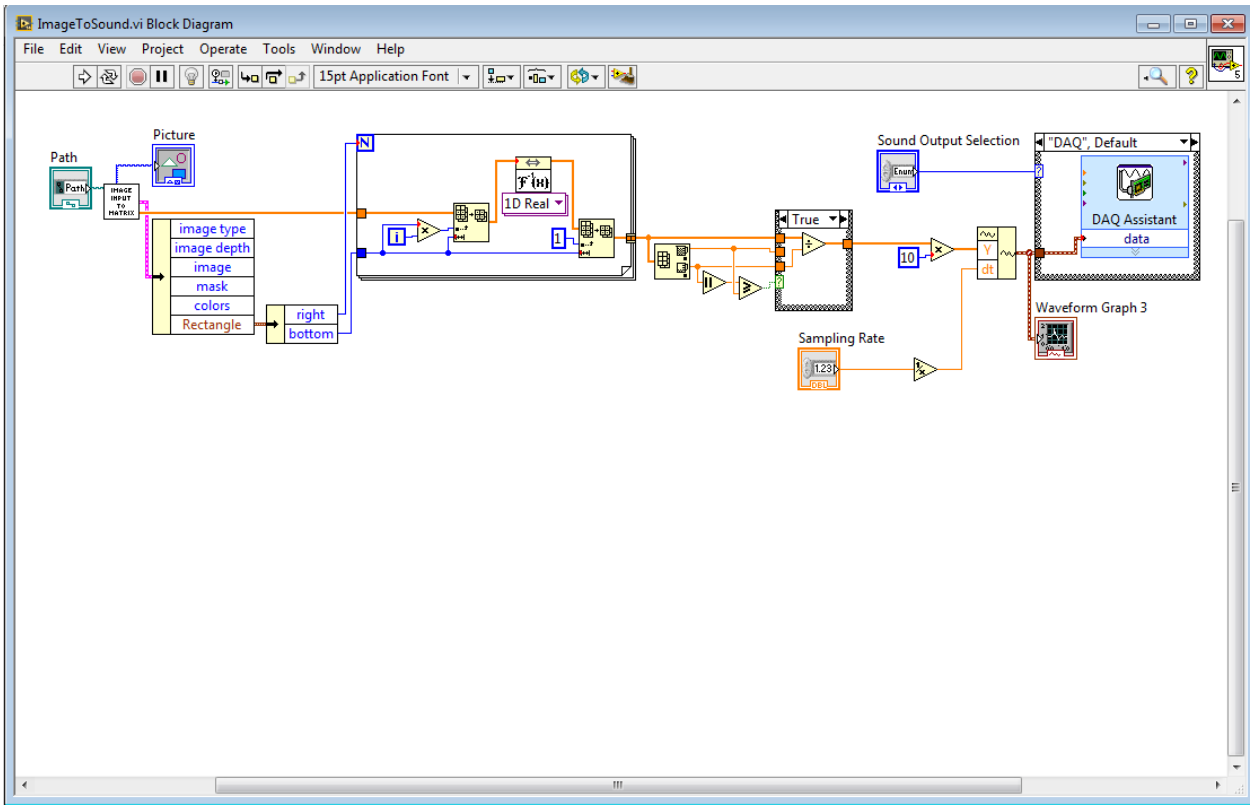
a. Overview of User Interface

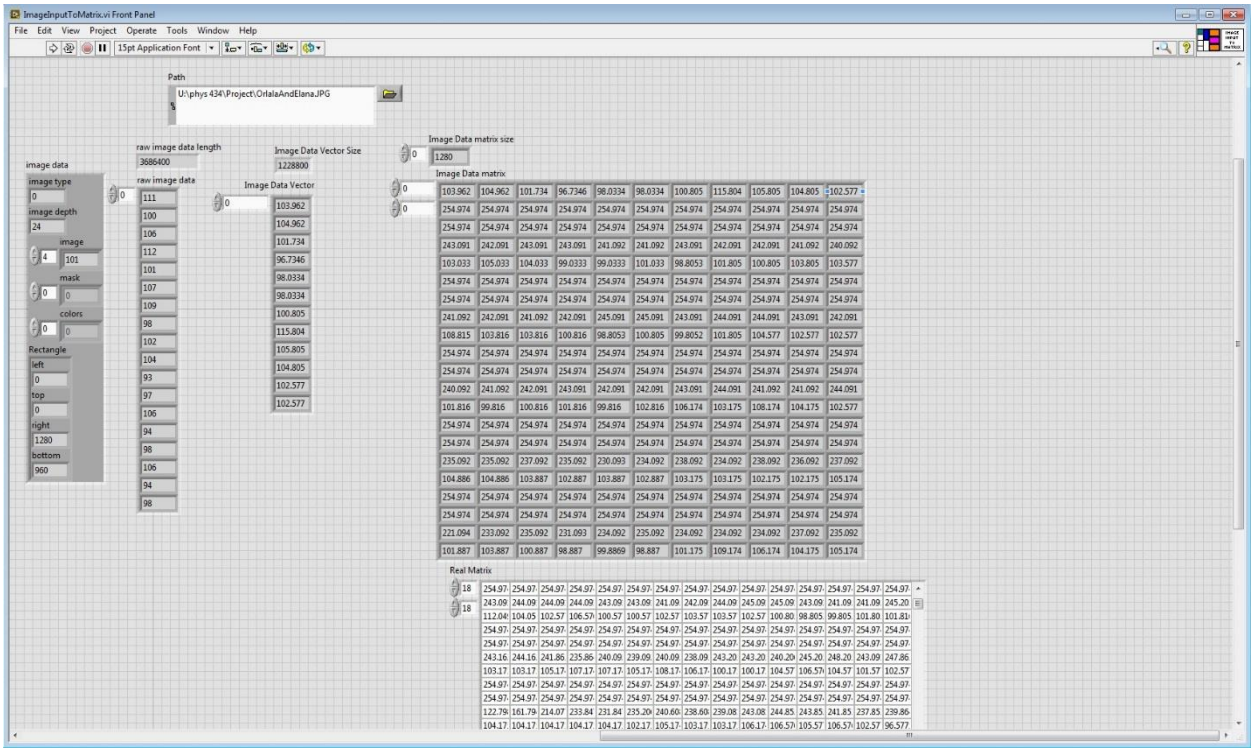
The user picks a JPEG image through file path interface and enters a sampling rate. The user must also define if the audio output should play through the DAQ or the computer speakers. The program displays the waveform from the IFFT and plays back an audio version of the signal, through output designated by the user. The user must also define the sampling rate. This is arbitrary for the purposes of simply converting a random image, the smaller the sampling rate the longer the sound produced. Recommended values for standard images are between 10000 and 100000 to produce a sound of reasonable length. However, higher and lower values can be selected and run, they will just produce extremely short or long sounds respectively; beyond 900500, and below 1x10^-50 you may experience DAQ failure.

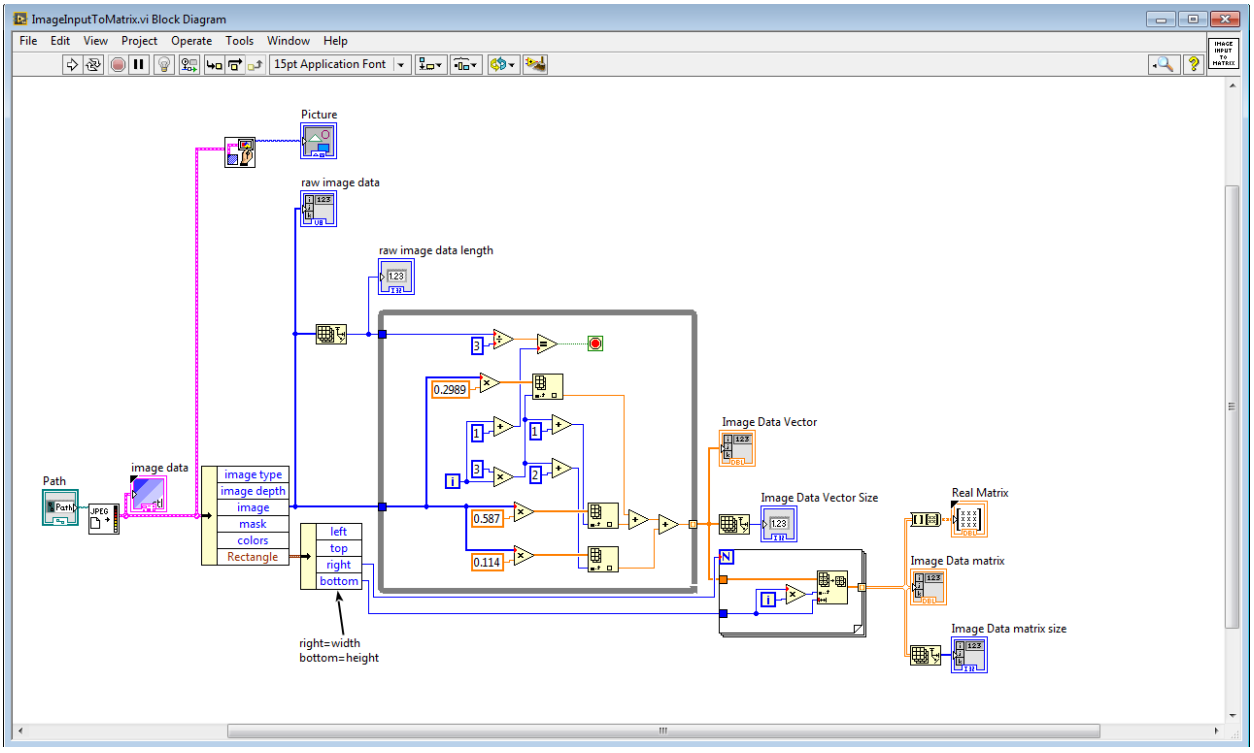
b. Screen Images

The following are our VI's ImageToSound and ImageInputToMatrix.

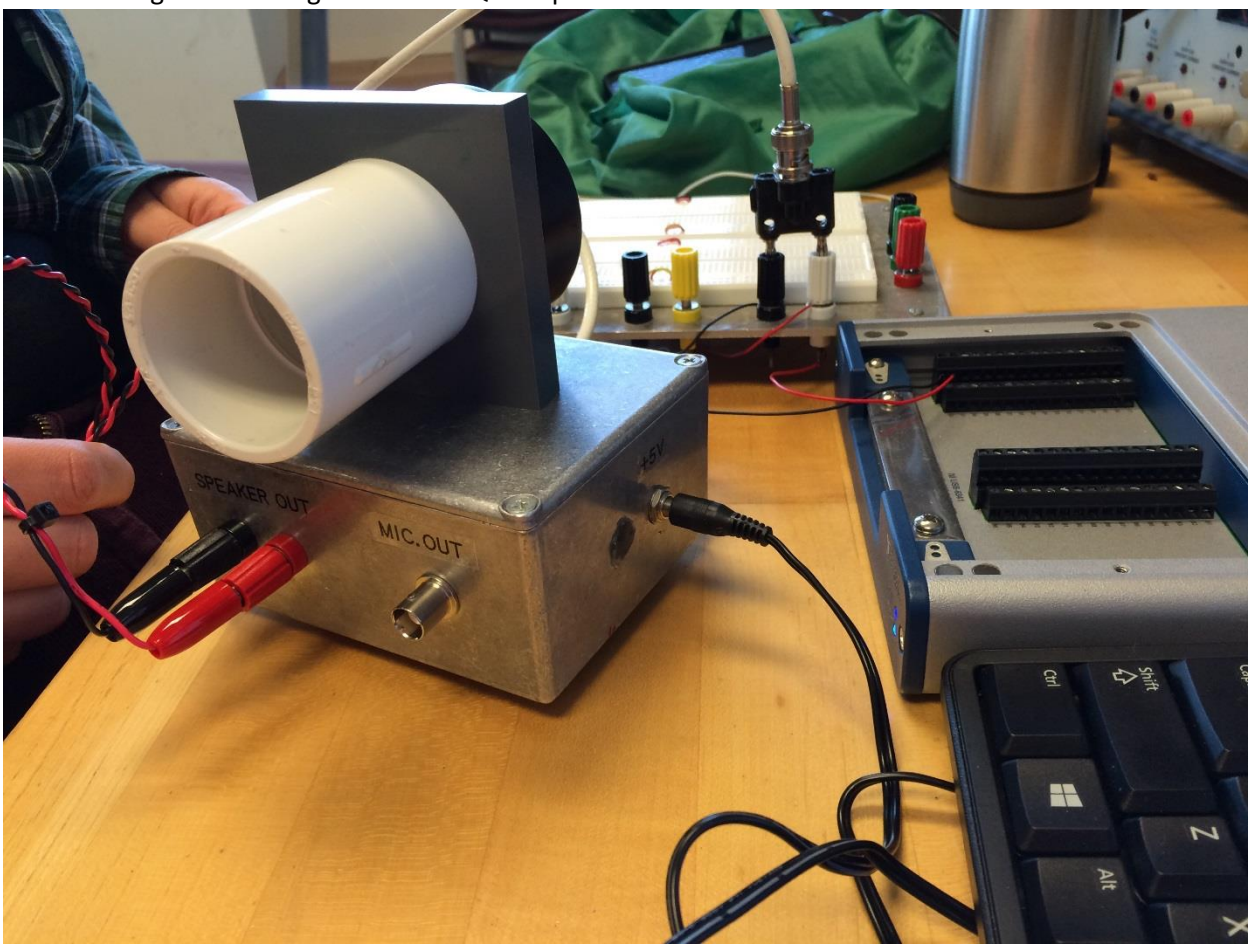

ImageToSound.vi Front Panel
File Edit View Project Operate Tools Window Help
15pt Application Font
Path
U:\phys 434\Project\OrlalaAndElana.JPG
Sound Output Selection
DAQ
Sampling Rate
50000
Waveform Graph 3
Plot 0
Amplitude
Time

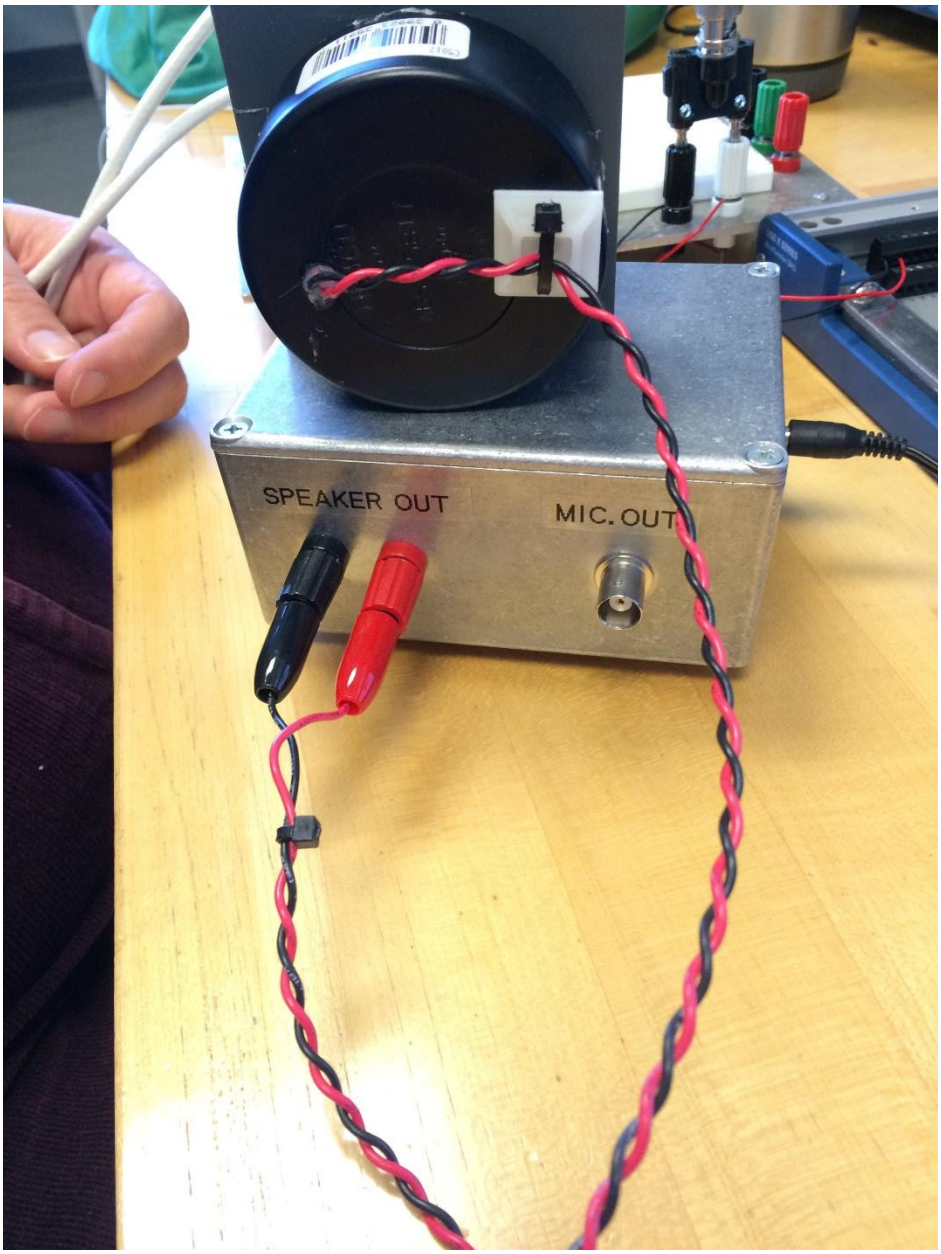

ImageToSound.vi Block Diagram
File Edit View Project Operate Tools Window Help
15pt Application Font
Path
Picture
image type
image depth
image
mask
colors
Rectangle
right
bottom
1D Real
True
Sound Output Selection
"DAQ", Default
DAQ Assistant
data
10
Waveform Graph 3
Sampling Rate

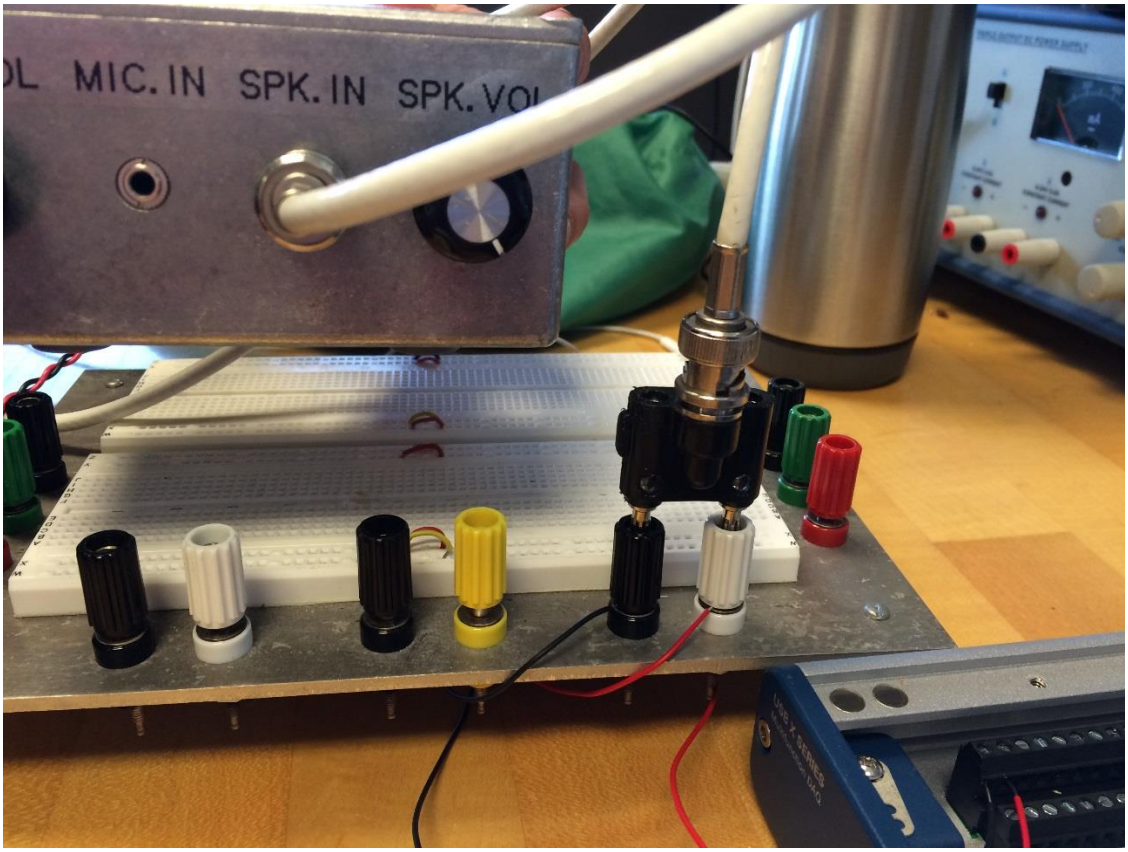

ImageInputToMatrix.vi Front Panel
File Edit View Project Operate Tools Window Help
15pt Application Font
Path
U:\phys 434\Project\OrlalaAndElana.JPG
image data
image type
image depth
image
mask
colors
Rectangle
left
top
right
bottom
raw image data length
raw image data
Image Data Vector Size
Image Data Vector
Image Data matrix size
Image Data matrix
Real Matrix

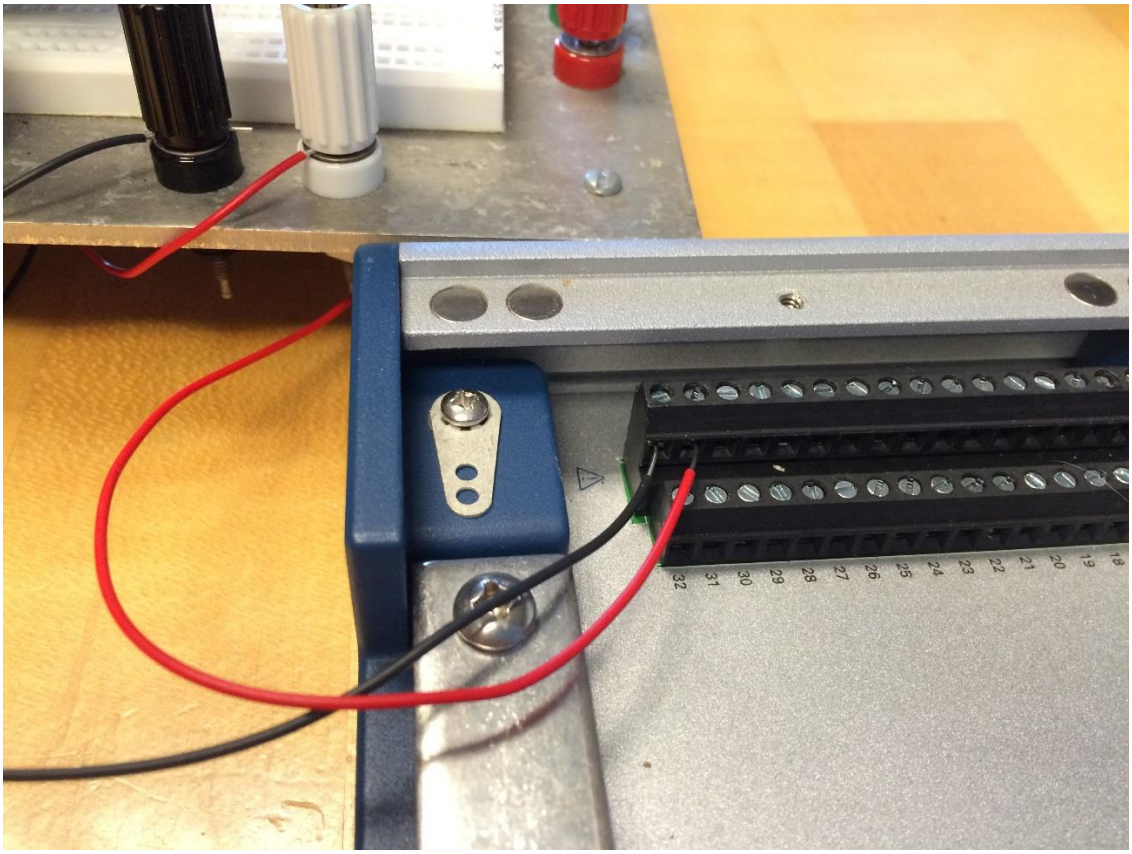

ImageInputToMatrix.vi Block Diagram
File Edit View Project Operate Tools Window Help
15pt Application Font
Picture
raw image data
raw image data length
Path
image data
image type
image depth
image
mask
colors
Rectangle
left
top
right
bottom
right=width
bottom=height
0.2989
0.587
0.114
Image Data Vector
Image Data Vector Size
Real Matrix
Image Data matrix
Image Data matrix size

The following are the images of our DAQ set up.




SPEAKER OUT
MIC. OUT


MIC. IN
SPK. IN



The image below is the spectrogram from a linguist 103 class that we used and referenced in 3b. We got the audio file related to this, which is in our project folder, from the same source.

Hayes, Bruce. Spectrogram Reading Answer 1. Department of Linguistics UCLA. Fall 2013. Accessed 12/12/2014. <http://www.linguistics.ucla.edu/people/hayes/103/SpectrogramReading/Answer1.htm>