Project Description

Our touch-tone recgnition project introduces a practical aplication of transmitting information via sinusoidal signals, and demonstrates the effects of encoding and decoding this information.  We implemented our touch-tone recognition project using Matlab, which consisted of 5 main parts.  These functions are core to our design and recognition algorithm, and include the following:

* Dtmfdial.m
* Dtmfcut.m
* Dtmfdesign.m
* Dtmfscore.m
* Dtmfrun.m

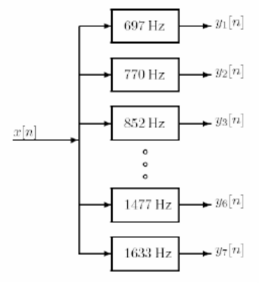
First, we generate our own touch-tones and DTMF signal by summing each pair of unique frequencies together in the Dtmfdial.m funciton.  Next, if the DTMF signal consists of multiple touch-tones, we parse the signal to obtain a set of individual tone segments in the Dtmfcut.m function. Then we put each tone through a series of bandpass filters and compute the impulse response to isolate individual sinusoids.  These bandpass filters are applied to extract the information encoded in the waveforms.  By taking the fast fourier transform (FFT) in Dtmfdesign.m, we effectively take the discrete DTMF signal in the time domain and transform it into its discrete frequency domain representation.  Working in the frequency domain allows us to analyze the signal’s spectrum and “filter” the signal for expected DTMF frequencies.

After filtering, we have Dtmfscore.m check if the input signal frequency matches frequencies within the filter passbands.  Essentially, it convolves the impulse response of a bandpass filter and a touch-tone, and if its magnitude is greater than a certain threshold value, it passes through the filter.  Finally, we decode and return a text string corresponding to the touch-tones represented by the DTMF signal in Dtmfrun.m.

Our project also involves more sophisticated touch-tone recognition processes.  We have enhanced our filters to implement noise generation and filtration.  And we have also explored processing touch-tones generated by a computer’s microphone (e.g. recording a cellular telephone’s touch-tone), and doing so in real-time.

**Filtering Noise**

The 5-part touch-tone recognition system described above needed to be modified when introducing random noise to the signal.  This is because the random noise heavily distorts the DTMF signal, causing frequency shifts and requiring a modified filter system to recognize touch-tones.  In the original filter system, the DTMF signal would traverse through a filter bank consisting of bandpass filters with center frequencies matching those of the eight DTMF component frequencies.  The new filter system is more robust, and is explained in detail below.

 Figure … Original filter bank design from Dtmfdesign.m

**Norman’s part**

Recognizing an Audio Input

To make our project more like a real telephone, we implemented functions to record, play, and recognize DTMF touch-tones generated from a cellular telephone using a microphone.

Wavrecord

Real time looping through

Processing recording

Putting it all Together (GUI)

We chose to create a graphical user interface based on specific objectives/goals pertaining to touch-tone recognition, including simulating a recognition system that could substitute for a real telephone receiver.  The GUI was essential to integrating all of the Matlab functions together, and allowed our project to be easily presentable during a demonstration.

The final GUI consisted of five main parts: a keypad, noise addition, alternate dialing methods, plots, and filters/DTMF recognition.  The keypad featured a standard DTMF/touch-tone layout, and allowed users to simulate the motion of dialing a telephone.  When a push button on a keypad was pressed, it generated the corresponding DTMF signal—storing the value into a global variable, playing the sound, and plotting the signal in the time domain.  With the GUI, adding noise to the signal was almost effortless.  A checkbox and slider were used to determine whether to add noise to the signal and how much noise to add.  This noise section of the GUI also featured the options to play the new signal, plot it, and clear the signal completely.  The alternate dialing methods section allowed users to generate touch-tones in new, unconventional ways—which included dialing a string of characters and dialing from the computer’s microphone.  Dialing a string employs the same method of touch-tone creation as the keypad, but does so in a for-loop for the length of the text string.  Dialing from the computer’s microphone is much more complicated, as it requires recording an analog audio input and using it as a DTMF signal.  The plots were useful in obtaining a visual grasp of the functions performed on the DTMF signal.  With these, we could view the signal in both the time-domain and the frequency-domain, and visually confirm the identity of the touch-tones.  The last part of the GUI involved the filtering and post processing of the DTMF input signal.  A “Filter” button allowed you to see the eight bandpass filters that were used to isolate individual frequencies.  There were also text fields, which displayed the DTMF number you dialed and the number our touch-tone recognizer decoded.

Hieu’s Individual Contribution

Because I have had more experience using Matlab, my main responsibility for this project was creating/programming the graphical user interface (GUI).  The GUI made the project easier to integrate our Matlab functions together, and made it easier to test/debug our code; instead of repeatedly entering commands via command line, we could now efficiently click and choose which functions to run.

I’ve never created a GUI in Matlab before, so it was challenging to learn it on my own.  Luckily, Matlab has a wide user-base, so there are several tutorials and examples available on the Internet.  Specifically, I had some trouble with correctly plotting the fast Fourier transform, sharing variables among different functions, and implementing the real-time recognition part of the project.  At first, when I plotted the fft of the DTMF signal, it would generate a symmetrical graph of the frequency spectrum, but the horizontal axis did not correctly correspond to the actual frequencies.  This was because I forgot to scale the independent variable by the sampling frequency at 8kHz.  I also isolated the left half of the fft, in order to process the important information.  Using global variables was essential to coding the GUI because several functions rely on the DTMF input signal, which carries a history of modifications (adding noise, different lengths, etc.).  I ended up using the handles structure in Matlab to adequately share variables between different Matlab functions of the GUI.  Implementing real-time recognition was difficult because it involved continuously looping through last-second microphone feedback and running our recognition algorithm.  Additionally, Matlab is not extremely fast so it was a challenge to get “real-time” recognition to work properly.

Another contribution I was responsible for was figuring out how to analyze DTMF signals using a recorded audio input.  I looked up some example code online and researched how to record audio from a computer microphone, format the useful data into a vector (to be used in our DTMF recognition code), and parse the DTMF sound file to recognize timing differences for other applications.  Ultimately, I was able to get this code to function properly using similar algorithms from the original/noisy touch-tone recognition.  However, this part was limited in that it could only recognize one touch-tone at a time.  This is because of the inconsistencies between tone bursts and silence; each recorded audio signal would be different in the time domain, and this made it difficult to parse.  If I had additional time, I believe I could solve this problem by writing code to segment the time signal based on amplitude differences, thus preserving the sequential order of the DTMF signal.

