DTMF Decoding Using Filter Banks and Modulation

Henry Troutman

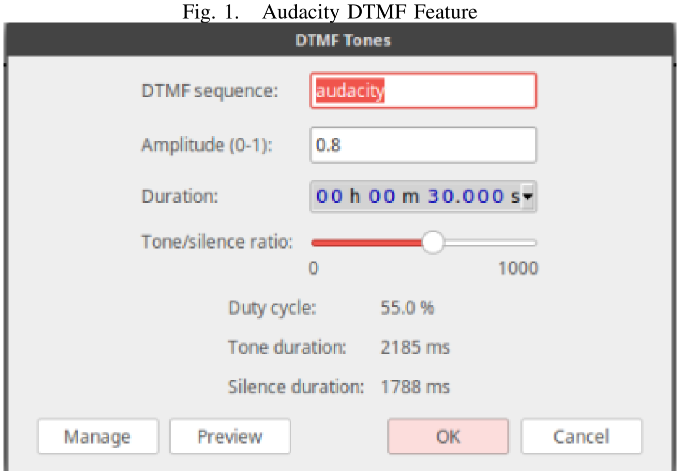
***Abstract*— Dual-tone multi-frequency signaling is a telecommunications standard that relies on mixing two sets of frequencies to encode numerical data on a telephone line. The process of decoding this information was a problem that was originally solved using analog signal processing, but it can easily be done using digital signal processing. There are several approaches to solving this problem but, two methods in particular were explored: Filter Banks, and Modulation. These two methods were implemented, and their efficacy was evaluated under ranging test conditions. The methods and their results are enumerated in this document.**

I. Implementation

A. Input

In the implementation of both the filter bank and modulation decoder, the input is a sound (wav) file with a sample rate of 8 kHz. Most of the files were generated using a sound edit software called *audacity*. Audacity has a feature for generating DTMF sequences, noise, and recording microphone audio. Additionally, the example files were used from Columbia University's Dan Ellis[1].

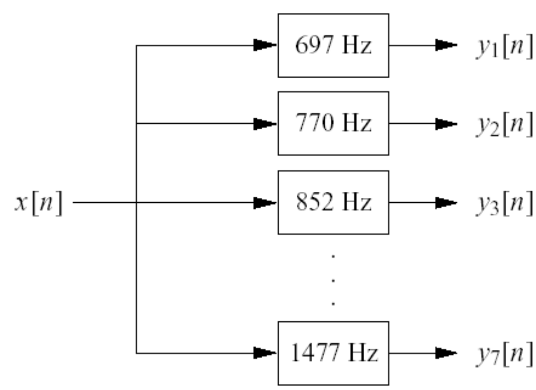
Fig. 1. Audacity DTMF Feature



B. Filter Bank

The filter bank approach is fairly straightforward. It requires implementing a band pass filter for each possible frequency in the DTMF scheme. The MATLAB filter design tool (fdatool) was used to create a butterworth band pass filter with an arbitrary cutoff frequencies. The filter was then exported to a matlab design file. The function was edited to make the cutoff frequencies parametric. In the main file, an array was created for each possible frequency. A *for* loop was used to loop through the array and create a filter for each frequency with a pass band of ± 25 Hz around the frequency. The data from the wav file is used as the input into each filter. The output of each filter is put into a matrix with a channel for each frequency *bin*. In order to condition the signal further, each channel was squared (to get the magnitude and remove negative components) and then smoothed using the moving mean function. Next, the algorithm must determine where each tone starts and ends. To do this, another loop was implemented that checks the current value against a threshold to see if the signal is active or silent at each point. Whenever it determines there is an active pulse, it will check which 2 channels of the filter bank has the greatest value (upper and lower frequencies). For every instance where a channel has a greater value, a count is incremented and when the pulse is finished the two frequencies with the greatest count are identified as the result. The result is then referenced with a table to get the symbol, and that symbol is added to the output sequence. This value is then returned from the program.

Fig. 2. Filter Bank Diagram



C. Modulation

The Modulation Method relies on the fact that modulating (multiplying) a tone by a sine wave of a known frequency will generate components of both the sum and the difference of the frequencies. Effectively, shifting the frequency range. When the input contains the same frequency as the modulation frequency, it will generate a DC component. By adding a filter after the modulation, the system essentially becomes a controllable filter. This can be useful if space

*This work was not supported by any organization
[1]H. Kwakernaak is with Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7500 AE Enschede, The Netherlands `h.kwakernaak at papercept.net`
[2]P. Misra is with the Department of Electrical Engineering, Wright State University, Dayton, OH 45435, USA `p.misra at ieee.org`

is an issue, because one filter can be used to check each frequency instead of a bank. However, it reduces the speed of the system because only one frequency can be checked at a time. For it to work, time has to be sliced up and the “controlled filter“ set to a different frequency at each slice. The first attempt was not successful because there wasn’t enough time during each pulse to check every frequency and wait for the delay of the system to propagate. This attempt was designed like figure 3. It was very space and memory efficient, but not fast. The method that ended up working much better was to modulate the input for each frequency individually. This system was much faster because it was parallel, however, it wasn’t as space efficient or as memory efficient. A diagram of this method is pictured in figure 4. The output of the filter and square function is a magnitude value, which is still far from the desired character sequence output. So, signal requires the same conditioning that filter bank had to determine where the pulses start and stop.

Fig. 3. Modulation with Multiplexing

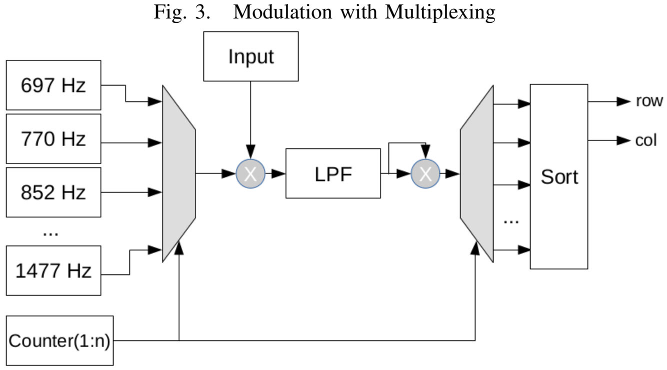
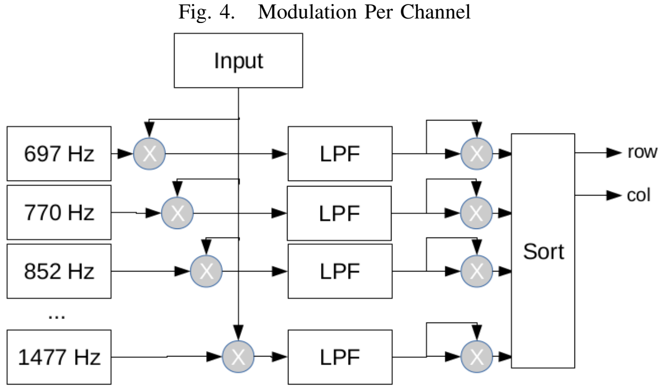


Fig. 4. Modulation Per Channel



II. Results

A. Filter Bank

The results of the filter bank show that it is relatively robust to noise. In the following test, the white noise was set to an amplitude of 0.5 and the DTMF tone was set to an amplitude of 0.8. Also, a 60Hz signal was added because 60Hz interference is a common real world problem. The algorithm responded with the correct result. The algorithm was also tested with the Professor Ellis’ files, and for every scheme it generated the correct result (Automatic dialing, hand-dialled, over-network, wideband-coupling).

Fig. 5. Input Spectrogram

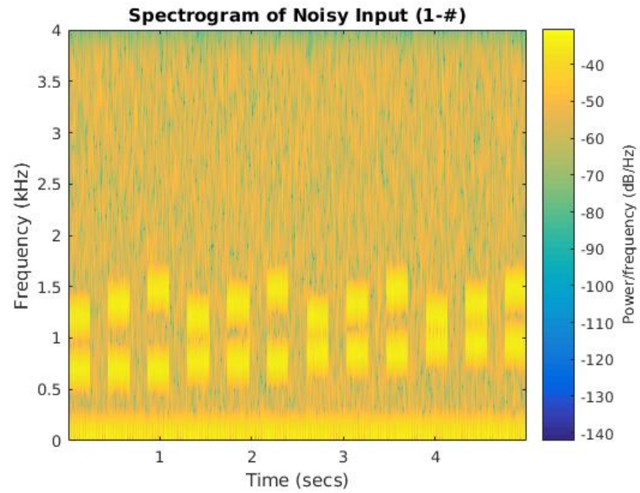


Fig. 6. Output of Filter Bank

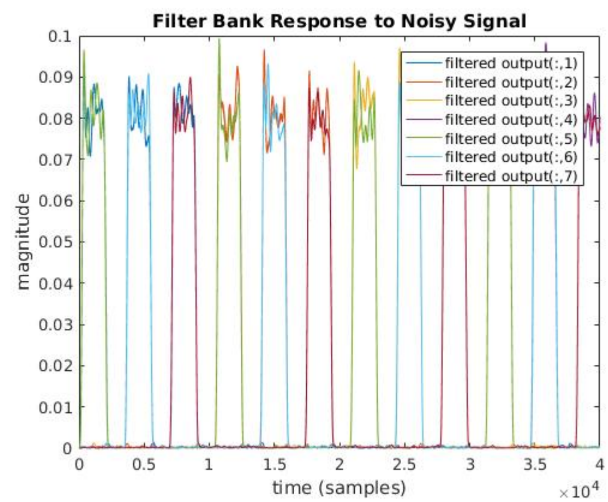


Fig. 7. Decoded Output Sequence

```
>> filterbank_decode
Decoded DTMF Sequence (Filter Bank):
123456789*0#
```

B. Modulation

The modulation algorithm was tested with the same test vectors as the filter bank and after adjusting it had the same success rate. From the plot it can be seen that with the same input vector, the output is sloppier. But it is still clear what is happening.

Fig. 8. Output of Modulation

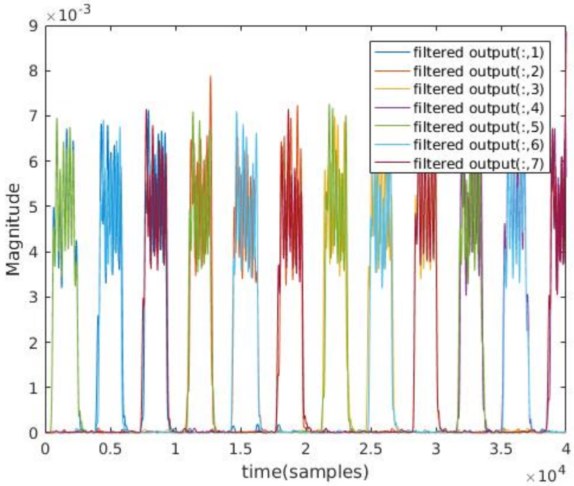


Fig. 9. Decoded Output Sequence

```
>> modulation_decode
Decoded DTMF Sequence (Modulation):
123456789*0#
```

III. Analysis

- The DTMF frequencies are specifically chosen to prevent the harmonics of the frequencies from overlapping with the primary values. Since the frequencies have no common multiple, the change of overlap is reduced.
- The algorithm has correctly decoded every digit and sequence that has been tested. Though it has been corrected, the algorithm has the most trouble differentiating the lowest two of the frequencies.
- When the keypress is longer, the filters have longer to propagate. Since there is a group delay determined by the order for each filter there is a limit to how fast they can respond. However, better filters can be designed with higher orders. So, speed is traded with accuracy by constraining the order of the filter.
- When increasing the noise in the input, the signal to noise ratio decreases. At some point the the ratio will be too low for any system to distinguish values. However, when using white noise it is distributed evenly in the spectrum. One of the advantages of this algorithm is that it compares the relative intensity of each bin, not absolute. So, the white noise has a lessened effect.
- As long as the noise isn't exactly the DTMF frequencies, the input can be filtered to remove noise. For example, if there is a 60Hz signal from the power lines in the building. A notch filter could be designed to remove that noise. The algorithm already combats noise by comparing the magnitudes of the DTMF frequencies between each other, instead of to an absolute threshold. Noise would raise all the values, but the relative difference between the bins would be less affected.
- The algorithm was tested with voice interference, and it was able to successfully identify the sequence. But it can be seen from the plot that there is more high frequency noise in each signal. Also the purple signal is much greater which could result in a false positive for the signal.

Fig. 10. Input with Voice Interference

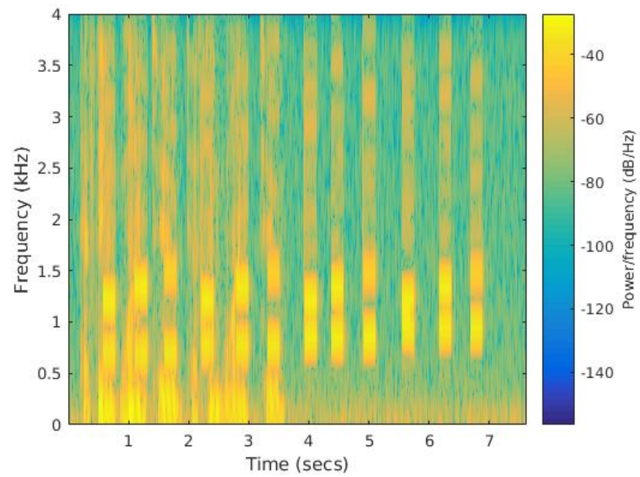
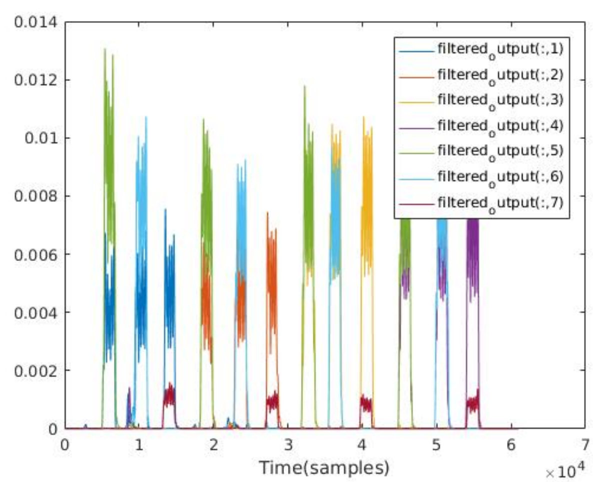


Fig. 11. Voice Interference Response



IV. Conclusion

In order to construct a DTMF decoder two methods were evaluated, a filter bank and modulation. The filter bank has the advantage of speed due to its parallel nature, but it suffers from using a lot of resources. The modulation strategy has the advantage of using less resources, but is slower. Both of these decoders were simulated using a variety of inputs. The noise, interference, and timing of DTMF sequences were varied and tested. The filter bank performed the best, but both had a nearly perfect success rate. These strategies are not just for DTMF decoding, but can be applied to many other signal processing problems. Solving this problem served as a beneficial learning experience.

APPENDIX

```matlab
% DSP Final Project (Filter bank)
% Author: Henry Troutman
filename = 'wavfiles/noise50p.wav';
freqs = [697,770,852,941,1209,1336,1477];
keys = ['1','2','3';'4','5','6';'7','8','9';'*','0','#'];
fs = 8000;
% This quantity represents what fraction of the highest value the signal
% needs to be to qualify as silence
silence_diff_threshold = 5;
% read the file
[y,Fs] = audioread(filename);
% downsample if necessary
if Fs ~= fs
    disp('Mismatching sample rate, looking for 8kHz');
    if(Fs>fs)
        disp('decimating down')
        decimate(y,Fs/fs);
    else
        disp('quiting');
        stop;
    end
end

figure(1);
spectrogram(y,50,25,2048,fs,'yaxis');
filtered_output = zeros(length(y),length(freqs));
% loop through each frequency and apply a bandpass filter
temp = zeros(length(y),1);
% ripple and stop gain
Rp = 3;
Rs = 40;
for i = 1:length(freqs)
    % Stop +- 25 Hz from the desired frequency
    Hd = myfilter(freqs(i)-25,freqs(i)+25);
    temp = filter(Hd,y);
    % remove negative component
    temp = temp.*temp;
    % smooth the wave with an average
    filtered_output(:,i) = movmean(temp,200);
end
figure(2);
plot(filtered_output,'DisplayName','filtered_output');
% scale the threshold for different volume files
```

```
silence_threshold = max(max(filtered_output))/silence_diff_threshold;
guesses = [];
% These will store a count of each occurance where the frequency
% is the greatest of the 7
total1 = zeros(4,1);
total2 = zeros(3,1);
% state is used to find the falling edge when silence is reached
state = 0;
% loop through the time data
for i = 1:length(filtered_output)
    % check the sum of every filter against the threshold
    if sum(filtered_output(i,:)) < silence_threshold
        % silence is found
        if state == 0
            % this is not the first frame of silence
            % reset the data to prepare for the next tone
            total1 = zeros(4,1);
            total2 = zeros(3,1);
        else
            % this is the first frame of silence
            % store the data from the previous tone
            [m,f1] = max(total1);
            [m,f2] = max(total2);
            guesses = [guesses,keys(f1,f2)];
            state=0;
        end

    else
        % Determine which filter channel has the greatest value
        % for the upper and lower bands
        [m,f1] = max(filtered_output(i,1:4));
        [m,f2] = max(filtered_output(i,5:end));
        % increment the total value for each frequency
        total1(f1)=total1(f1)+1;
        total2(f2)=total2(f2)+1;
        % change the state to indicate that there is data to
        % be stored
        state = 1;
```

```
        % Check if this is the last frame of the sound file
        if i==length(filtered_output)-1
            % store the final data since there may not be silence at
            % the end of the file
            [m,f1] = max(total1);
            [m,f2] = max(total2);
            guesses = [guesses,keys(f1,f2)];
            state=0;
        end
    end
end
% Print the results
disp('Decoded DTMF Sequence (Filter Bank):');
disp(guesses);
```

```
% DSP Final Project (modulation)
% Author: Henry Troutman
filename = 'wavfiles/voice.wav';
freqs = 2.*pi.*([697,770,852,941,1209,1336,1477]);
keys = ['1','2','3';'4','5','6';'7','8','9';'*','0','#'];
fs = 8000;
% This quantity represents what fraction of the highest value the signal
% needs to be to qualify as silence
silence_diff_threshold = 5;
% read the file
[y,Fs] = audioread(filename);
% downsample if necessary
if Fs ~= fs
    disp('Mismatching sample rate, looking for 8kHz');
    if(Fs>fs)
        disp('decimating down')
        decimate(y,Fs/fs);
    else
        disp('quiting');
        stop;
    end
end

figure(1);
```

```
spectrogram(y,50,25,2048,fs,'yaxis');
% load an empty array
filtered_output = zeros(length(y),length(freqs));
% t values for the sin function at the sample rate
sine_domain = 0:1/fs:(length(y)/fs-1/fs);
temp = zeros(length(y),1);
% loop through each frequency and apply modulation and a filter
for i = 1:length(freqs)
    % When using the exact frequency, the output ends up
    % getting clipped below DC so +100 is added to keep it at dc
    modulation = 0.5*cos((freqs(i)+100).*sine_domain);
    signal_sum = y'.*modulation;
    % filter gets us closer to dc
    Hd = lp_mod_filter2();
    temp = filter(Hd,signal_sum);
    % remove negative component
    temp = temp.*temp;
    % use the movmean to get the DC offset
    filtered_output(:,i) = movmean(temp,200);
end
figure(2);
plot(filtered_output,'DisplayName','filtered_output');
% scale the threshold for different volume files
silence_threshold = max(max(filtered_output))/silence_diff_threshold;
guesses = [];
% These will store a count of each occurance where the frequency
% is the greatest of the 7
total1 = zeros(4,1);
total2 = zeros(3,1);
% state is used to find the falling edge when silence is reached
state = 0;
% loop through the time data
for i = 1:length(filtered_output)
    % check the sum of every filter against the threshold
    if sum(filtered_output(i,:)) < silence_threshold
        % silence is found
        if state == 0
            % this is not the first frame of silence
```

```
            % reset the data to prepare for the next tone
            total1 = zeros(4,1);
            total2 = zeros(3,1);
        else
            % this is the first frame of silence
            % store the data from the previous tone
            [m,f1] = max(total1);
            [m,f2] = max(total2);
            guesses = [guesses,keys(f1,f2)];
            state=0;
        end

    else
        % Determine which filter channel has the greatest value
        % for the upper and lower bands
        [m,f1] = max(filtered_output(i,1:4));
        [m,f2] = max(filtered_output(i,5:end));
        % increment the total value for each frequency
        total1(f1)=total1(f1)+1;
        total2(f2)=total2(f2)+1;
        % change the state to indicate that there is data to
        % be stored
        state = 1;
        % Check if this is the last frame of the sound file
        if i==length(filtered_output)-1
            % store the final data since there may not be silence at
            % the end of the file
            [m,f1] = max(total1);
            [m,f2] = max(total2);
            guesses = [guesses,keys(f1,f2)];
            state=0;
        end
    end
end
% Print the results
disp('Decoded DTMF Sequence (Modulation):');
disp(guesses);
```

```matlab
function Hd = myfilter(Fc1,Fc2)
%MYFILTER Returns a discrete-time filter object.

% MATLAB Code
% Generated by MATLAB(R) 9.1 and the DSP System Toolbox 9.3.
% Generated on: 09-Dec-2018 18:46:05

% Butterworth Bandpass filter designed using FDESIGN.BANDPASS.

% All frequency values are in Hz.
Fs = 8000;  % Sampling Frequency

N   = 10;   % Order

% Construct an FDESIGN object and call its BUTTER method.
h  = fdesign.bandpass('N,F3dB1,F3dB2', N, Fc1, Fc2, Fs);
Hd = design(h, 'butter');

% [EOF]
```

```matlab
function Hd = lp_mod_filter2
%LP_MOD_FILTER2 Returns a discrete-time filter object.

% MATLAB Code
% Generated by MATLAB(R) 9.1 and the DSP System Toolbox 9.3.
% Generated on: 10-Dec-2018 00:10:57

% Butterworth Lowpass filter designed using FDESIGN.LOWPASS.

% All frequency values are in Hz.
Fs = 8000;  % Sampling Frequency

Fpass = 30;          % Passband Frequency
Fstop = 50;          % Stopband Frequency
Apass = 1;           % Passband Ripple (dB)
Astop = 80;          % Stopband Attenuation (dB)
match = 'stopband';  % Band to match exactly

% Construct an FDESIGN object and call its BUTTER method.
```

```
h  = fdesign.lowpass(Fpass, Fstop, Apass, Astop, Fs);
Hd = design(h, 'butter', 'MatchExactly', match);

% [EOF]
```

References

[1] JD. Ellis, DTMF examples, About Colored Noise. [Online]. Available: http://www.ee.columbia.edu/ dpwe/sounds/dtmf/. [Accessed: 08-Dec-2018].