DTMF Decoder

Design and implementation of a DTMF Decoder

Tinotenda Chemvura

The problem

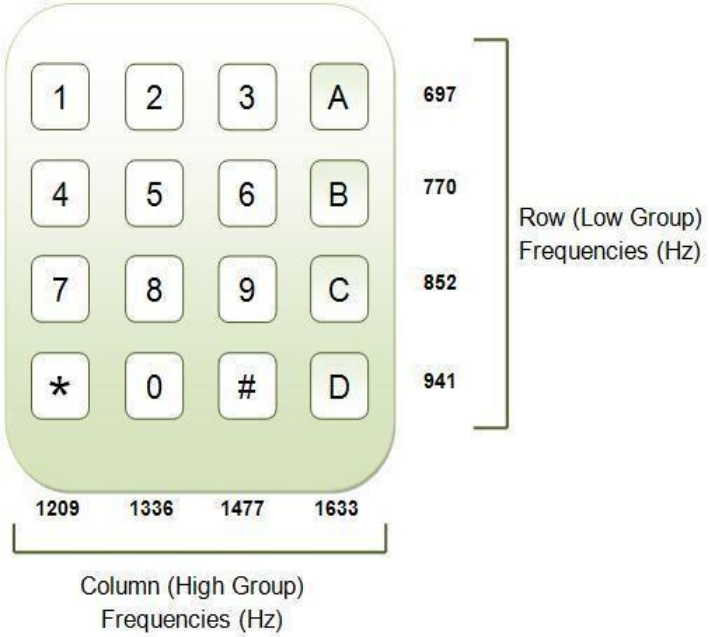
What is DTMF

DTMF - Dual Tone Multi-Frequency

Developed by Bell in 1950, first used in 1963.

Represents 16 keys

Usage: Telecoms, automated locking/unlocking, controlling robots, etc...

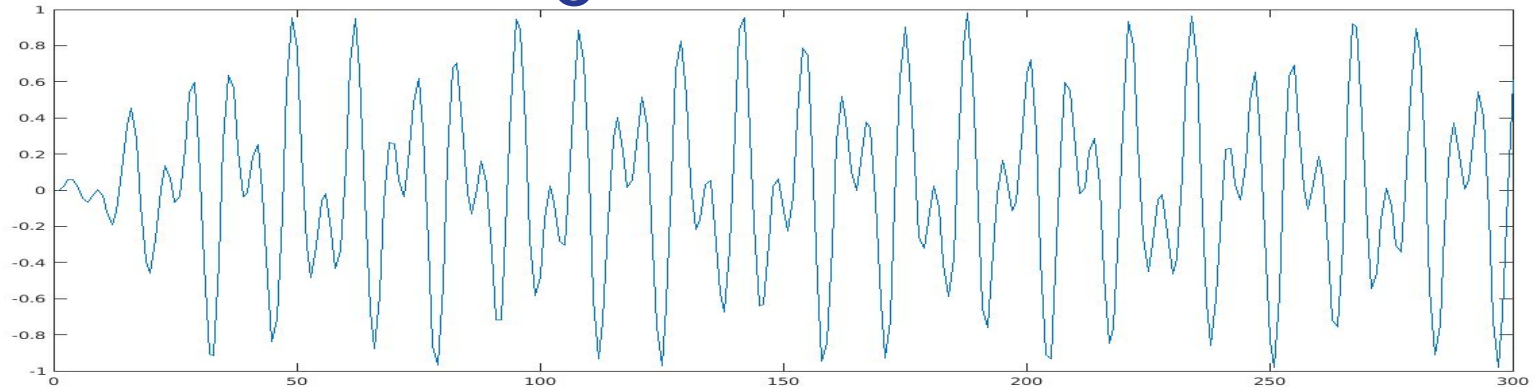


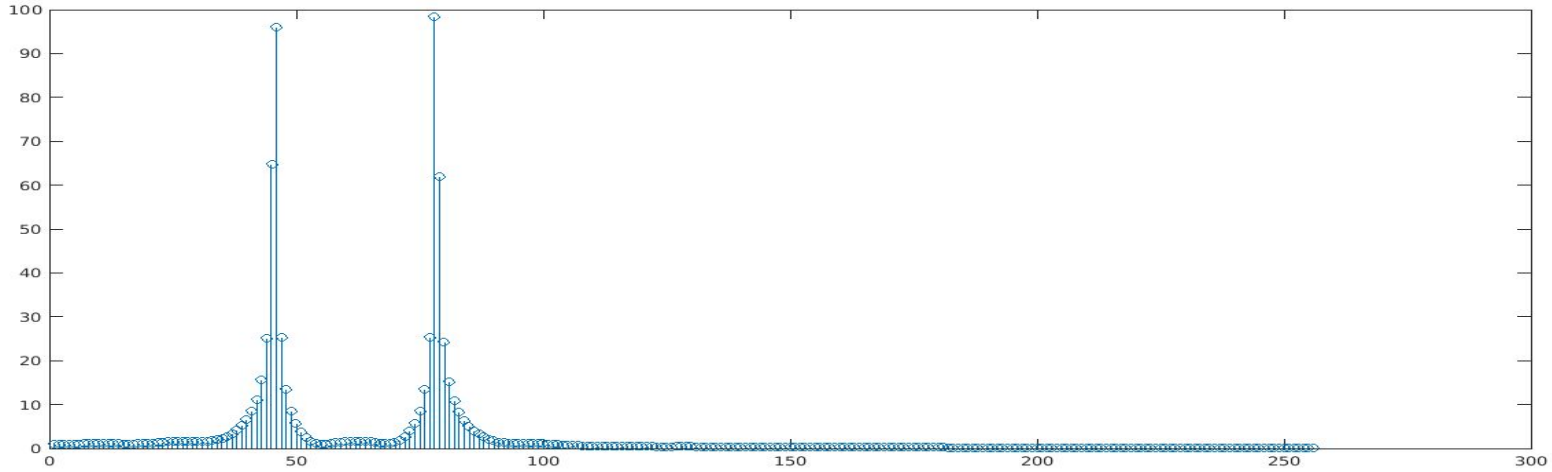
Problem statement

Design a DTMF Decoder that follows ITU-T Recommendations and implement it in Java.

Background and Research

Research and Background





Research and Background

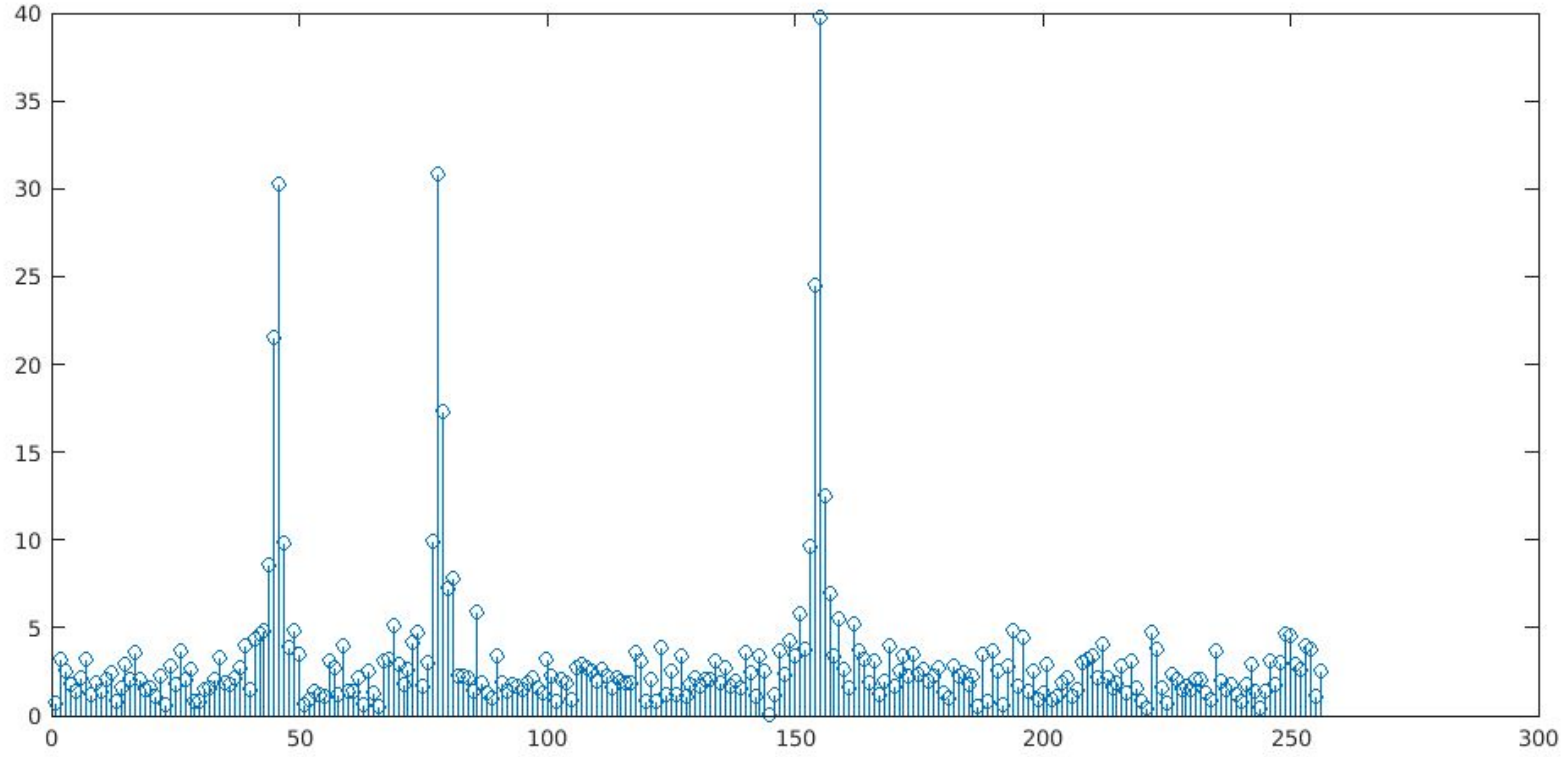
Fast Fourier Transform

- DFT algorithm developed by Cooley and Tukey (1965).
- Use it to get the power spectrum of the whole signal.
- Better option for noise rejection and eliminating voice and other non-DTMF signals.
- Lots of computations therefore slow.

Goertzel's Algorithm

- DFT algorithm developed by Gerald Goertzel (1958).
- Used to get DFT data of individual terms of a DFT.
- Higher complexity than FFT but much faster when computing few frequencies
- Not ideal for detecting signals with background noise consisting of non-DTMF tones and.

Research and Background



Prototyping

Test Data

- ❏ Generated using MatLab.
- ❏ ITU-T Recommendations (Q23 and Q24)
  - Signal Frequencies:
    - Low Band (Hz): 697, 770, 852, 941
    - High Band (Hz): 1209, 1336, 1477, 1633
  - Frequency Tolerances:
    - Max. accepted frequency offset: <=1.5%
    - Min. rejected frequency offset: >= 3.5%
  - Power Levels per Frequency:
    - Power range: -27dBm to 0dBm
    - Maximum twist: 5dBm
  - Signal Timing:
    - Min. accepted tone duration: 40ms
    - Min. pause: 30ms to 70ms
    - Max. rejected tone duration: 20ms
  - Max. signal interruption: 10ms

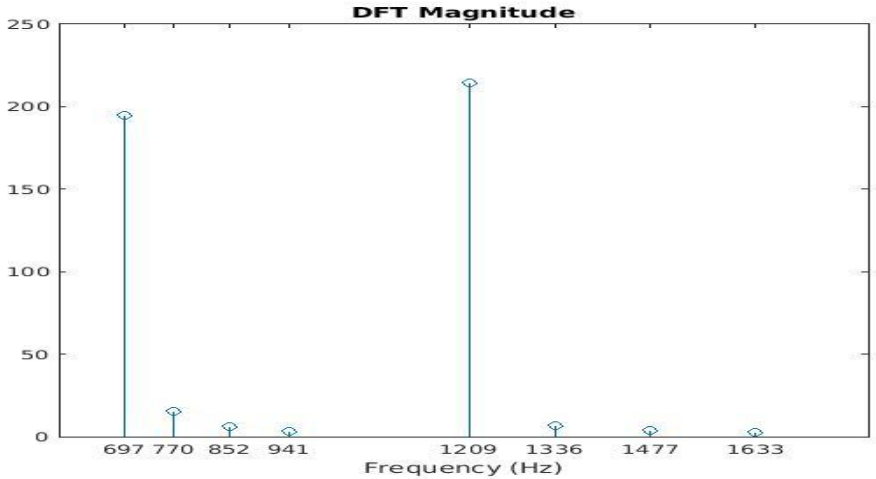
- The sequence of tones will be randomly determined (choosing from the 16 available DTMF characters) and the length of the sequence will also vary randomly from 5 tones to 50 tones per file.
- The duration of each DTMF tone will be varied randomly between 40ms and 50ms.
- The duration of the pause between DTMF tones will be randomly varied between 30ms and 50ms.
- To vary the power of the signal between 0dBm and -27dBm, the amplitude of the signal will be varied from 0.045 (-27dBm) to 1.0 (0dBm).
- Each tone will have 10ms white noise in a random position
- Tones will be generated using a randomly chosen frequency within the offset limits (<=1.5%)

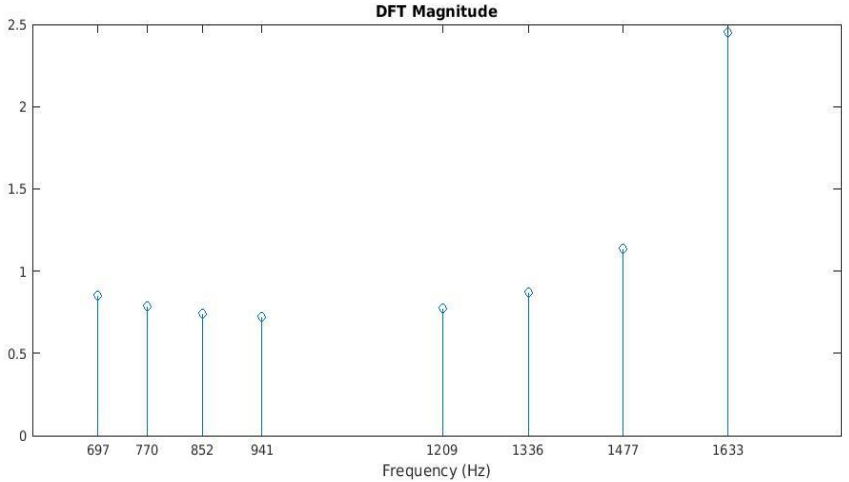
Prototyping

Done in MatLab

Algorithm [Goertzel Approach]:

- Split input into short frames (bins). Size depends on Fs but at least 32ms. Frames overlap by 33%.
- Check for signal power and reject if too low.
- Transform the frames and return power spectrum data.
- Get ratio of sum of two highest peaks from each bin to sum off 8 frequencies and compare to a predetermined ratio.
- If ratio high enough, determine character by identifying the DTMF frequencies present.

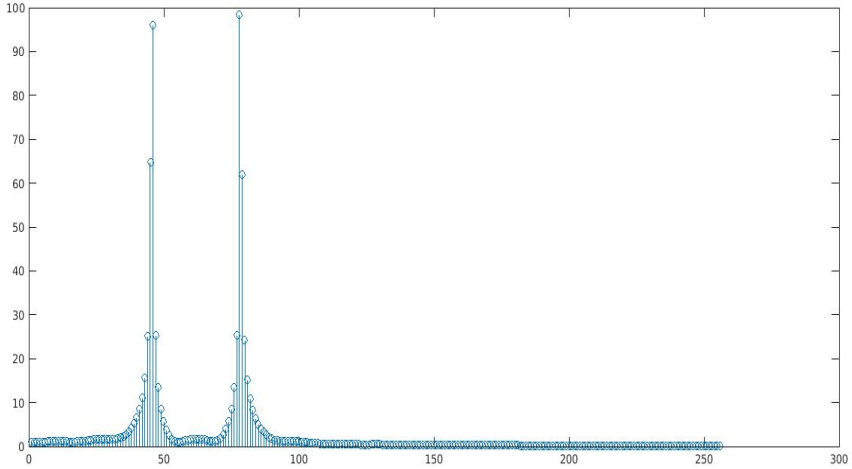


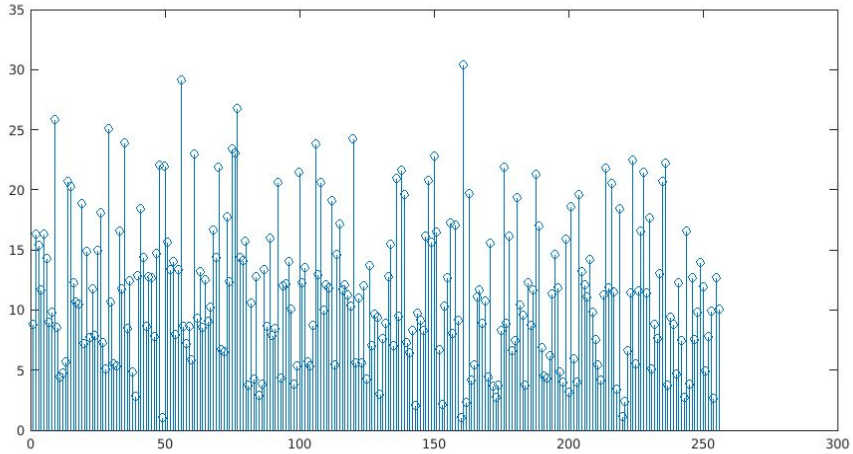


Prototyping

Algorithm [FFT Approach]:

- Split input into short frames (bins). Size depends on Fs but at least 32ms. Frames overlap by 33%.
- Check for signal power and reject if too low.
- Transform the frames and return power spectrum data.
- Get ratio of sum of two highest peaks from each bin to sum all frequencies and compare to a predetermined ratio.
- If ratio high enough, determine character by identifying the DTMF frequencies present.





Implementation & Testing

Java

- Implemented same algorithm in Java.
- Had to create custom Goertzel Class for the Goertzel’s Algorithm approach.
- Created an API which can be used to decode an audio file or an array of sample points.
- Same API could generate DTMF tones.
- Used Apache Commons Math Library for FFT function.

Testing

pass: the whole sequence is successfully decoded

hit rate : percentage of the number of tones picked up by the decoder to the number of tones present.

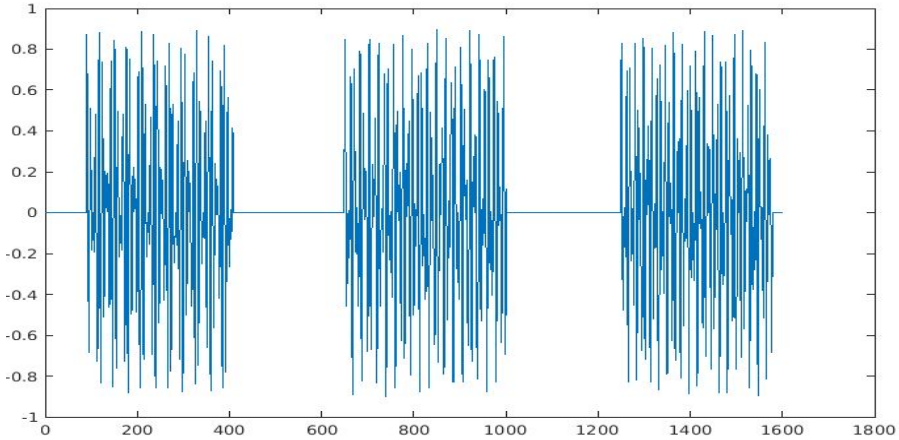
40,000 test files

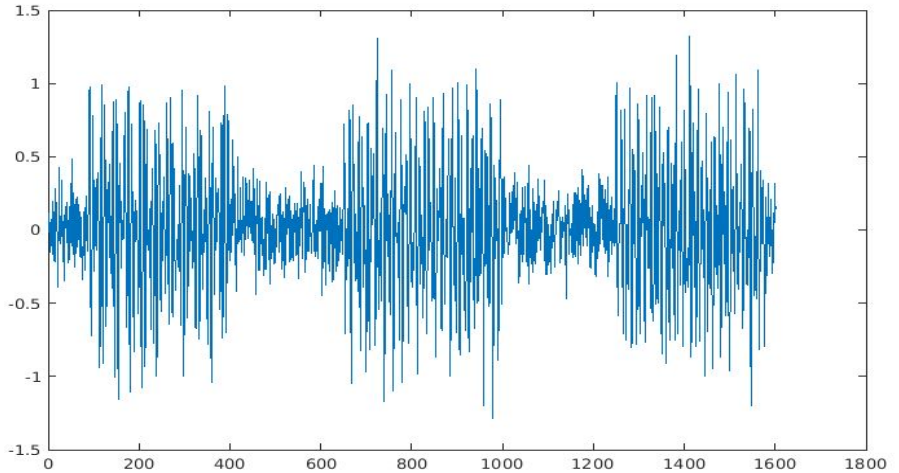
98.5% FFT pass Rate (99.4% Goertzel)

99.7% Hit Rate

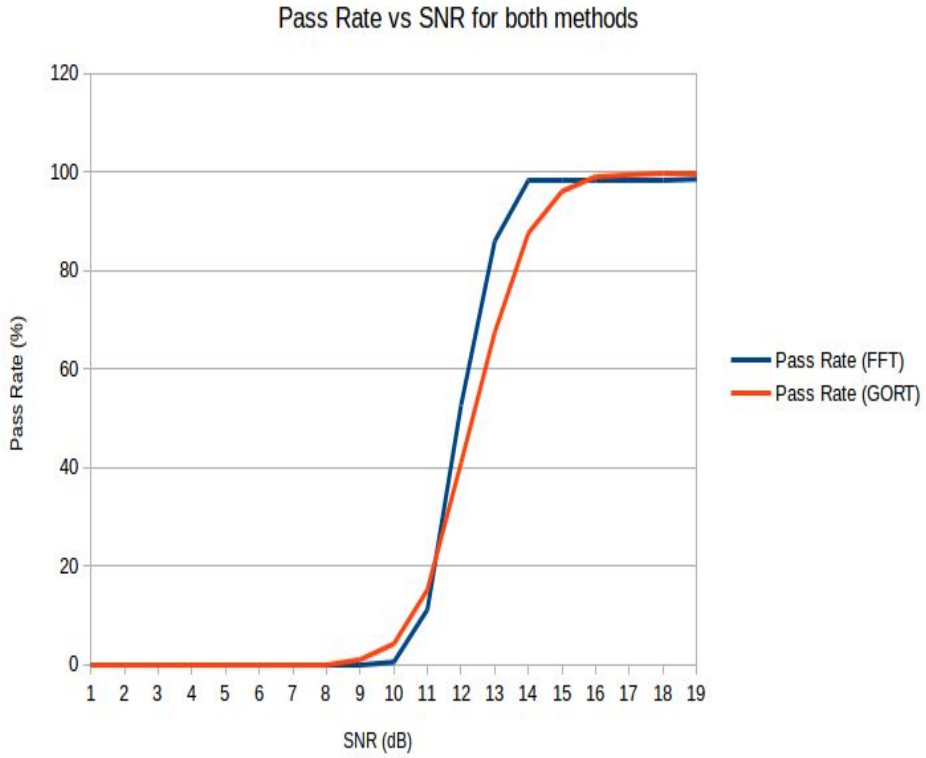
98.5% pass rate at 15dB (same trend for both methods)

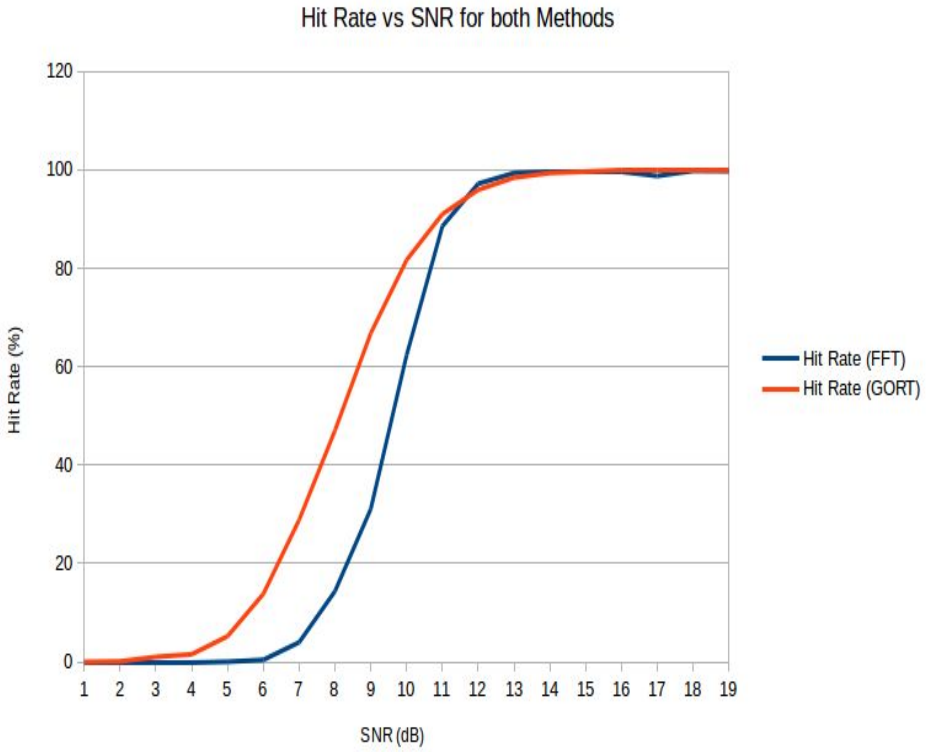
Better hit rate with Goertzel approach (WHY?)





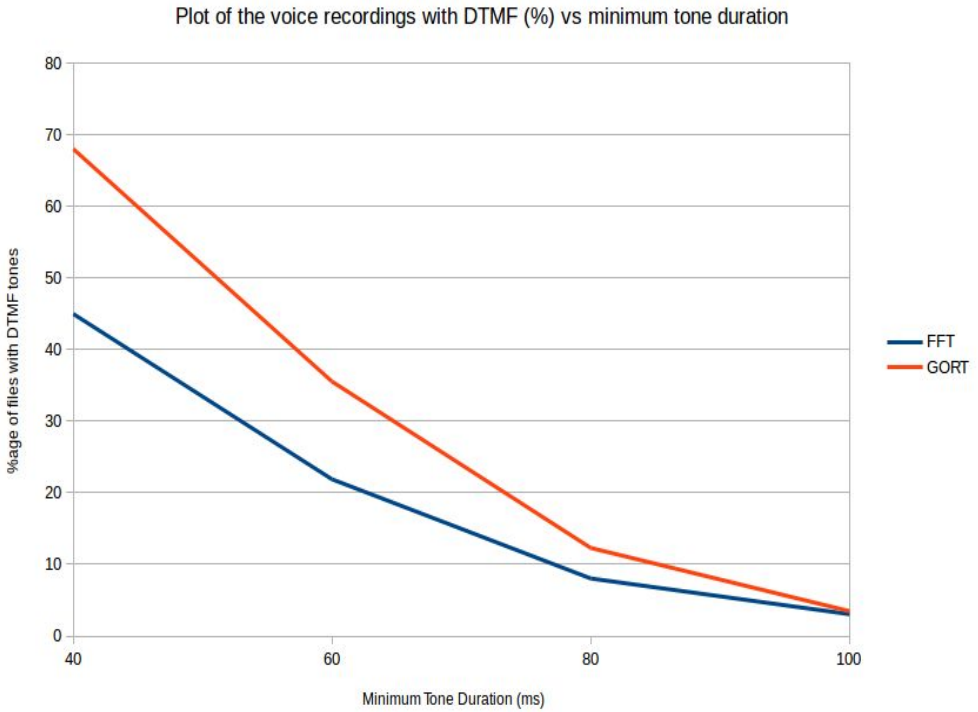
Testing


Pass Rate vs SNR for both methods
120
100
80
60
40
20
0
Pass Rate (%)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
SNR (dB)
Pass Rate (FFT)
Pass Rate (GORT)


Hit Rate vs SNR for both Methods
120
100
80
60
40
20
0
Hit Rate (%)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
SNR (dB)
Hit Rate (FFT)
Hit Rate (GORT)

Noise and Speech Testing

- Ran tests on voice recordings (total of 15,000 voice recordings). Very few of them had DTMF but no possible way of finding out which ones had DTMF and which ones did not.
- Goertzel method reported that 68% of the files had DTMF in them and FFT method reported 40%. This is where the FFT had a better advantage.
- Most were false hits caused by background music, files will background music and dial up connection sounds.
- The number of hits decreases as the minimum duration of a tone is increased...as shown in the graph.



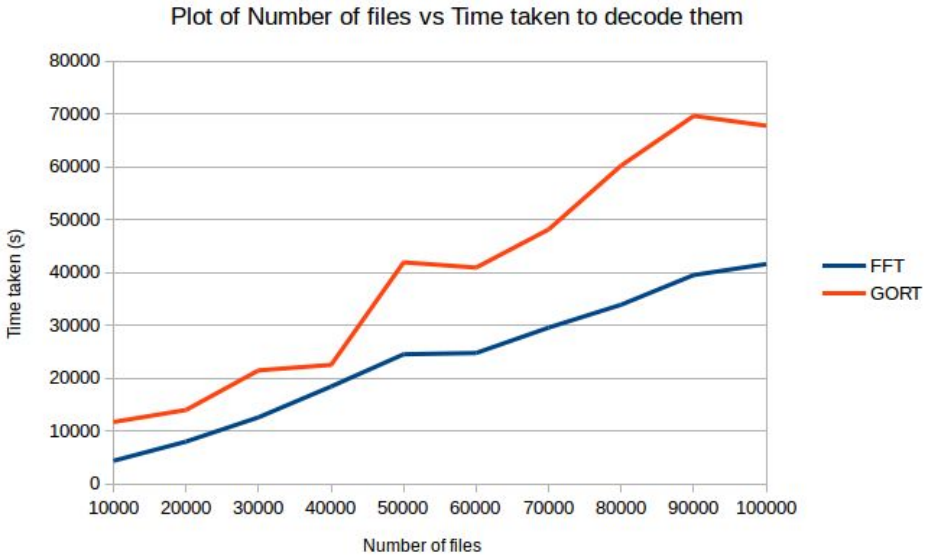
Goertzel vs FFT

Goertzel's Algorithm has higher degree of complexity but faster than FFT for a few target frequencies.

Ran timed tests of different number of files from 10,000 to 100,000 to compare the times taken by the two different algorithms to decode the files.

FFT performed better despite earlier assumptions that Goertzel's Algorithm would be faster.

Goertzel's Algorithm was not optimised. FFT from Apache Commons Math is optimised and is continuously updated.



Conclusions

Conclusion

- Project is on GitHub **@tino1b2be**
- Created an app to demonstrate the API.
- **API Specs:**
  - DTMF Decoder for .wav and .mp3 files and when given an array of sample points.
  - Has an audio file interface which can be implemented for more audio file types (ogg, wma, etc...)
  - DTMF Tone/Sequence Generator that can export to .wav files.
  - Goertzel Class which can be used independently
- Lots of room for improvement.

Possible Improvements:

- A better approach to rejecting music and speech. The detection ratio as it is used in the program is not very efficient.
- Optimise the Goertzel Class to for better performance.
- Decoder could give location of detected tones within the audio file.
-