DTMF TONE DECODER USING MATLAB

DETECTING TELEPHONE KEYPAD TONES VIA FREQUENCY ANALYSIS IN MATLAB

PRESENTED BY:

PEARL VYAS – 23BEC043
UMANG JHA – 23BEC049

UNDER MENTORSHIP OF :
DR. MANISH MANDLOI
DEPARTMENT OF ELECTRONICS AND COMMUNICATION,
PDEU, GANDHINAGAR

INTRODUCTION

- **What is DTMF?**

  Dual Tone Multi Frequency signaling is the method used by telephones to signal which key has been pressed using a combination of two tones—one low and one high frequency.

- **Why DTMF Decoding?**

  It's a practical application of signal processing, used in IVRS systems, telephone banking, call routing, etc.

- **Objectives of our Project:**
  - **Develop a real-time MATLAB system that:**
    - Captures audio from a microphone.
    - Detects valid DTMF tones.
    - Decodes them into digits of a phone number.
    - Displays the complete number after a defined silence period.

UNDERSTANDING THE PROBLEM STATEMENT

Problem Statement:
"To build a MATLAB-based system that decodes DTMF tones captured through a microphone and reconstructs a phone number."

Challenges Involved:

Real-time signal capture and processing.
Accurate tone detection amid noise.
Reliable digit recognition with time constraints.
Detecting end of input using silence detection.

Expected Outcome:
A working tool that can detect a sequence of keypad presses and output a complete phone number.

CODE

```
disp('DTMF Tone Decoder - Single Phone Number Detection');
disp('Please tap on numbers when it shows waiting for DTMF tones');

% Low frequencies (row)
fLow = [697 770 852 941];
% High frequencies (column)
fHigh = [1209 1336 1477 1633];

% DTMF keypad matrix
keypad = ['1', '2', '3', 'A';
          '4', '5', '6', 'B';
          '7', '8', '9', 'C';
          '*', '0', '#', 'D'];

% Audio parameters
fs = 16000;                 % Sampling frequency (Hz)
frameSize = 2048;           % Analysis frame size
overlapSize = 1024;         % Overlap between frames
recordDuration = 30;        % Maximum recording duration (seconds)
toneThresholdFactor = 8;    % Dynamic threshold factor (multiplier above mean)
freqTolerance = 20;         % Frequency matching tolerance (Hz)
minToneDuration = 0.1;      % Minimum duration for a valid tone (seconds)
digitSilenceDuration = 0.1; % Minimum silence between individual digits (seconds)
endSilenceDuration = 2.0;   % Silence duration to consider number complete (seconds)

% Initialize audio recording
audioRecorder = audiorecorder(fs, 16, 1);
bufferSize = fs * recordDuration;
currentPhoneNumber = '';
lastDetectedDigit = '';
consecutiveFramesWithSameDigit = 0;
```

```
lastDetectedDigit = '';
consecutiveFramesWithSameDigit = 0;
silenceFrames = 0;
endSilenceFrames = 0;
minConsecutiveFrames = ceil(minToneDuration * fs / (frameSize - overlapSize));
minDigitSilenceFrames = ceil(digitSilenceDuration * fs / (frameSize - overlapSize));
endSilenceThreshold = ceil(endSilenceDuration * fs / (frameSize - overlapSize));
isRecordingDigit = false;
isReadyForNewNumber = true;
phoneNumberDetected = false;


% Start recording
disp('Recording started. Press Ctrl+C to stop.');
disp('Waiting for DTMF tones...');
record(audioRecorder);

try
    frameIndex = 1;
    while frameIndex + frameSize <= bufferSize && ~phoneNumberDetected

        while audioRecorder.get('CurrentSample') < frameIndex + frameSize - 1
            pause(0.01);
        end

        % Get the latest audio frame
        audioData = getaudiodata(audioRecorder);
        currentFrame = audioData(max(1, end-frameSize+1):end);

        % Apply window to reduce spectral leakage
        windowedFrame = currentFrame .* hamming(frameSize);
```

```
% Apply window to reduce spectral leakage
windowedFrame = currentFrame .* hamming(frameSize);

% Calculate the spectrum
N = length(windowedFrame);
X = abs(fft(windowedFrame))/N;
f = (0:N-1)*(fs/N);

% Only look at the first half (Nyquist limit)
X = X(1:floor(N/2));
f = f(1:floor(N/2));

% Set dynamic threshold based on the signal
dynamicThreshold = mean(X) * toneThresholdFactor;

% Implement safe peak finding to avoid warnings
if max(X) > dynamicThreshold
    % Find peaks in the spectrum using dynamic threshold
    [peaks, locs] = findpeaks(X, 'MinPeakHeight', dynamicThreshold, 'SortStr', 'descend');

    % Extract the frequencies of peaks
    detectedFreqs = f(locs);
else
    % If no peaks above threshold, set empty arrays
    peaks = [];
    locs = [];
    detectedFreqs = [];
end

% Reset end silence counter if we're starting a new number
if isReadyForNewNumber && length(peaks) >= 2
```

CODE

```matlab
% Reset end silence counter if we're starting a new number
if isReadyForNewNumber && length(peaks) >= 2
    isReadyForNewNumber = false;
    endSilenceFrames = 0;
end

% Check if we have at least two peaks
if length(peaks) >= 2
    % Look for a low and high frequency match
    lowFreqMatch = false;
    highFreqMatch = false;
    rowIndex = 0;
    colIndex = 0;

    for i = 1:length(detectedFreqs)
        % Check if this frequency matches a low frequency
        for r = 1:length(fLow)
            if abs(detectedFreqs(i) - fLow(r)) <= freqTolerance && ~lowFreqMatch
                lowFreqMatch = true;
                rowIndex = r;
                break;
            end
        end

        % Check if this frequency matches a high frequency
        for c = 1:length(fHigh)
            if abs(detectedFreqs(i) - fHigh(c)) <= freqTolerance && ~highFreqMatch
                highFreqMatch = true;
                colIndex = c;
                break;
            end
```

```matlab
        % If we found both frequencies, we can stop searching
        if lowFreqMatch && highFreqMatch
            break;
        end
    end

    % If we found a valid DTMF tone
    if lowFreqMatch && highFreqMatch
        currentDigit = keypad(rowIndex, colIndex);
        endSilenceFrames = 0; %

        % Handle consecutive frames with the same digit
        if currentDigit == lastDetectedDigit
            consecutiveFramesWithSameDigit = consecutiveFramesWithSameDigit + 1;
            silenceFrames = 0;

            % Register a new digit if we've seen it for long enough
            if consecutiveFramesWithSameDigit >= minConsecutiveFrames && ~isRecordingDigit
                currentPhoneNumber = [currentPhoneNumber currentDigit];
                fprintf('Digit detected: %s\n', currentDigit);
                isRecordingDigit = true;
            end
        else
            consecutiveFramesWithSameDigit = 1;
            lastDetectedDigit = currentDigit;
            silenceFrames = 0;
        end
    else
        % No valid DTMF tone detected in this frame (still frequencies but not DTMF)
```

CODE

```matlab
        end
    else
        % No valid DTMF tone detected in this frame (still frequencies but not DTMF)
        silenceFrames = silenceFrames + 1;
        endSilenceFrames = endSilenceFrames + 1;
        consecutiveFramesWithSameDigit = 0;

        if silenceFrames >= minDigitSilenceFrames
            isRecordingDigit = false;
        end
    end
else
    % Not enough peaks, count as silence
    silenceFrames = silenceFrames + 1;
    endSilenceFrames = endSilenceFrames + 1;
    consecutiveFramesWithSameDigit = 0;

    if silenceFrames >= minDigitSilenceFrames
        isRecordingDigit = false;
    end
end


% Check if we have a complete phone number (long silence after some digits)
if ~isReadyForNewNumber && endSilenceFrames >= endSilenceThreshold && ~isempty(currentPhoneNumber)
    disp( '===================================================');
    disp(['COMPLETE PHONE NUMBER DETECTED: ' currentPhoneNumber]);
    disp( '===================================================');
    phoneNumberDetected = true;  % Set flag to exit the loop
end
```

```matlab
            phoneNumberDetected = true;  % Set flag to exit the loop
        end

        % Move to the next frame with overlap
        frameIndex = frameIndex + (frameSize - overlapSize);
    end

    % Stop recording when a phone number is detected or maximum duration reached
    stop(audioRecorder);

    if ~phoneNumberDetected && ~isempty(currentPhoneNumber)
        disp(['Partial number detected: ' currentPhoneNumber]);
    elseif ~phoneNumberDetected
        disp('No digits were detected.');
    end

catch
    % Stop recording if interrupted
    stop(audioRecorder);
    disp('Recording stopped.');

    % Display any partially detected number
    if ~isempty(currentPhoneNumber)
        disp(['Partial number detected: ' currentPhoneNumber]);
    end
end
```

OUTPUT

Live Editor - C:\Users\Admin\Downloads\DSP LAB\DTFM_project.mlx *

DTFM_project.mlx *

OS_OA.mlx

```
DTMF Tone Decoder - Single Phone Number Detection

Please tap on numbers when it shows waiting for DTMF tones

Recording started. Press Ctrl+C to stop.

Waiting for DTMF tones...

Digit detected: 8
Digit detected: 4
Digit detected: 3
Digit detected: 1
Digit detected: 9
Digit detected: 5
Digit detected: 6
Digit detected: 7
Digit detected: 2
Digit detected: 8
====================================================
COMPLETE PHONE NUMBER DETECTED: 8431956728
====================================================
```

ALGORITHMS & METHODS USED

DTMF FREQUENCY PAIRS:

- LOW: 697, 770, 852, 941 HZ
- HIGH: 1209, 1336, 1477, 1633 HZ
- COMBINED IN A 4X4 MATRIX KEYPAD.

SIGNAL ACQUISITION:

- USING AUDIO RECORDER IN MATLAB.
- SAMPLE RATE: 16 KHZ
- FRAME SIZE: 2048 SAMPLES WITH 1024 OVERLAP.

TONE DETECTION (FFT + PEAK MATCHING):

- HAMMING WINDOW APPLIED TO REDUCE SPECTRAL LEAKAGE.
- FFT APPLIED TO EACH FRAME.
- DYNAMIC THRESHOLD SET AS MEAN(SIGNAL) * FACTOR.
- PEAKS EXTRACTED USING FIND PEAKS.

- FREQUENCY MATCHING:
- MATCH TWO PEAKS TO NEAREST DTMF PAIR WITH ±20 HZ TOLERANCE.
- DETECT DIGIT FROM KEYPAD MATRIX.

DIGIT VALIDATION & DEBOUNCE LOGIC:

- MINIMUM TONE DURATION REQUIRED FOR VALID DIGIT.
- SILENCE THRESHOLD USED TO DETECT END OF NUMBER INPUT.

RESULTS AND KEY FINDINGS

System Performance :

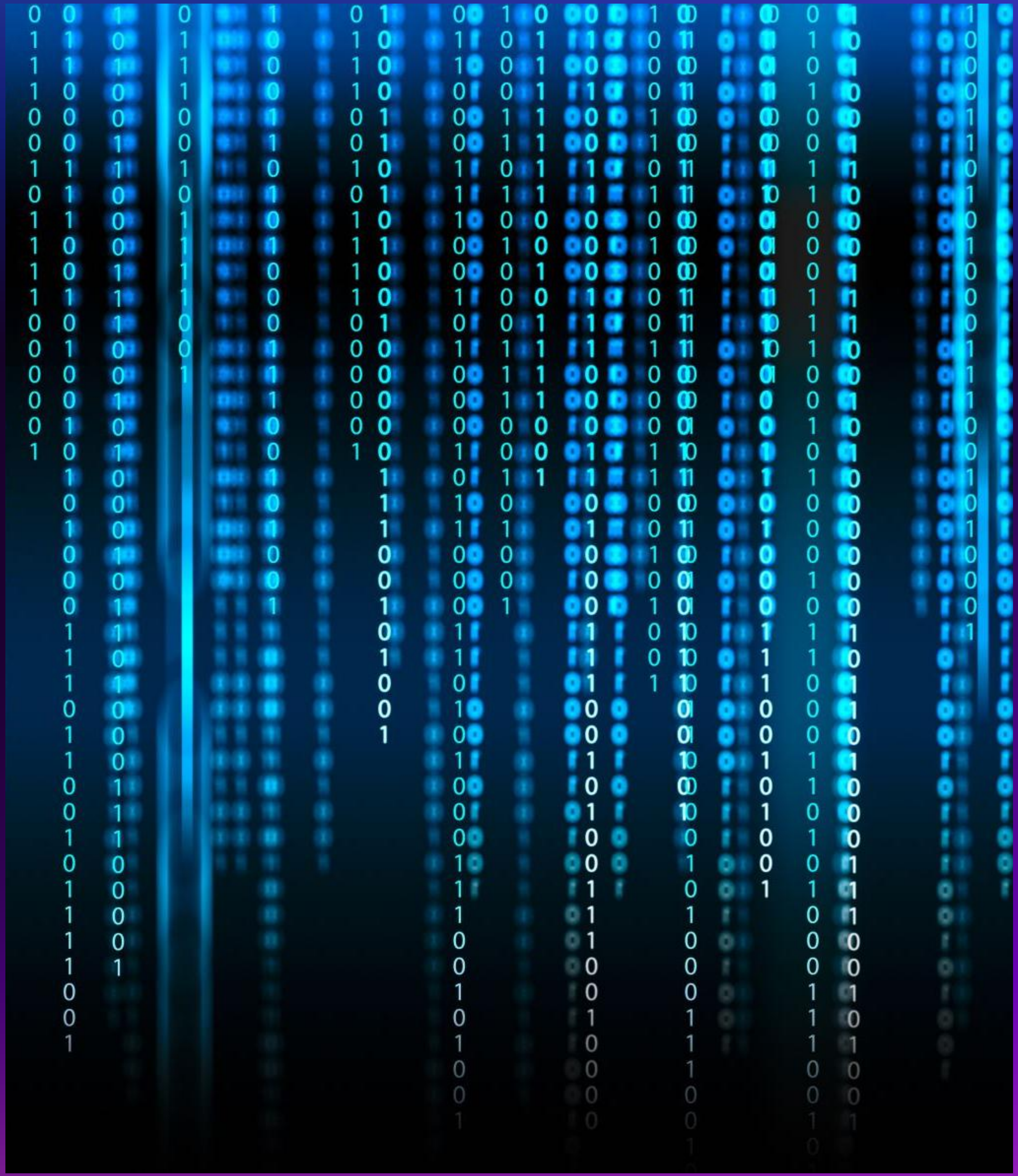
- Successfully detects digits from keypad tones in quiet environments.
- Capable of outputting entire phone number when input ends.
- Reliable distinction between digits using frequency matching.

Visual Output:

- Console prints each detected digit.
- Displays complete phone number after silence > 2 seconds.

Key Learnings:

- Real-time signal processing needs fine-tuning of timing and thresholds.
- FFT-based frequency detection is efficient for tone decoding.
- Noise handling and multi-frame logic significantly improve robustness.



CONCLUSION

A fully functional MATLAB-based DTMF decoder was developed that captures audio signals, detects DTMF tones, and reconstructs a complete phone number reliably.

Future Improvements:

- Add GuI (GRAPHICAL USER INTERFACE) for better interactivity.
- Implement machine learning for tone classification.
- Improve noise filtering for field use.

Applications:

IVR systems, access control, phone-based authentication.