Rahmaan Lodhia

GTID: 902502749

ECE 4271

Project #1

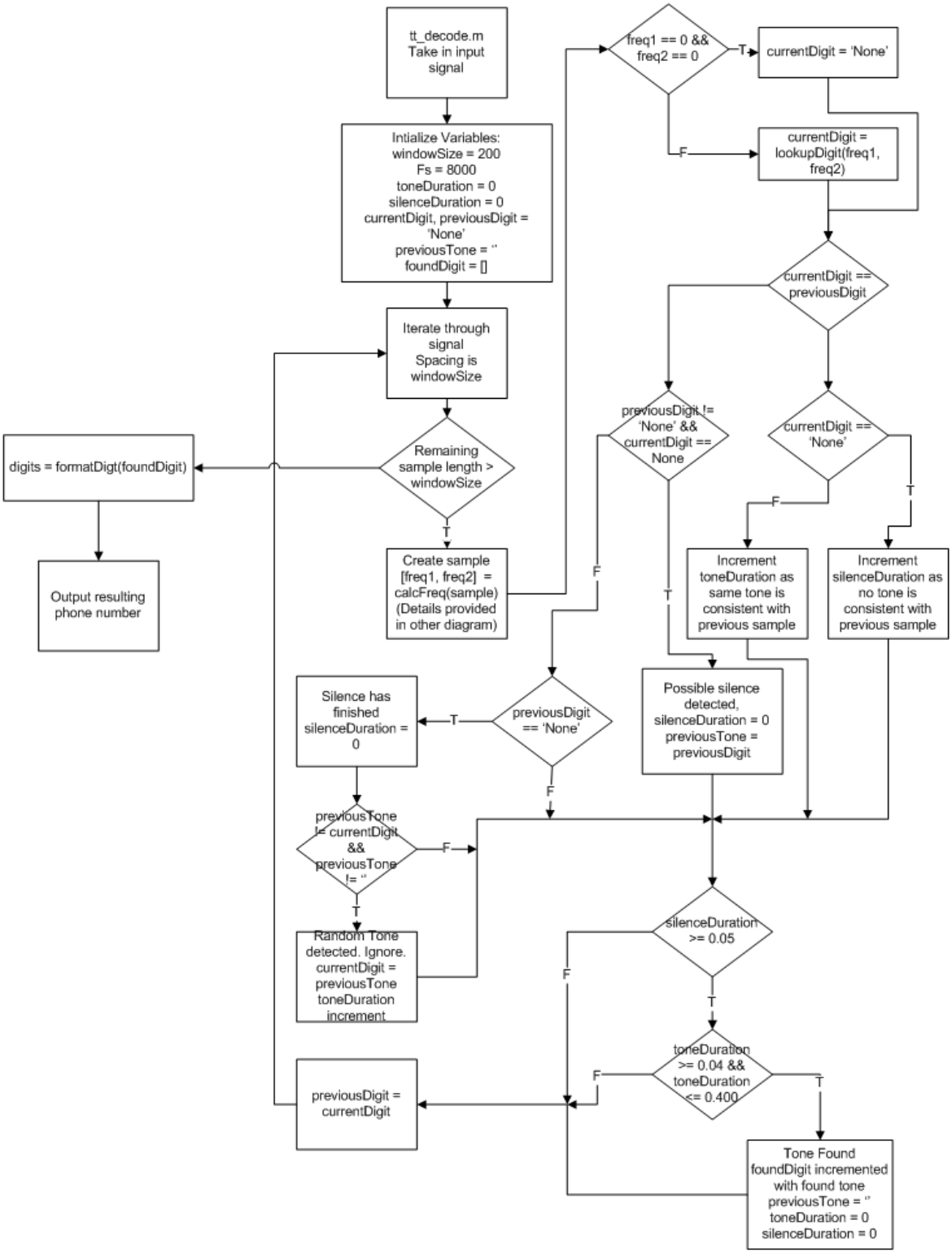
DTMF Signal Decoding

1. DTMF Algorithm Overview

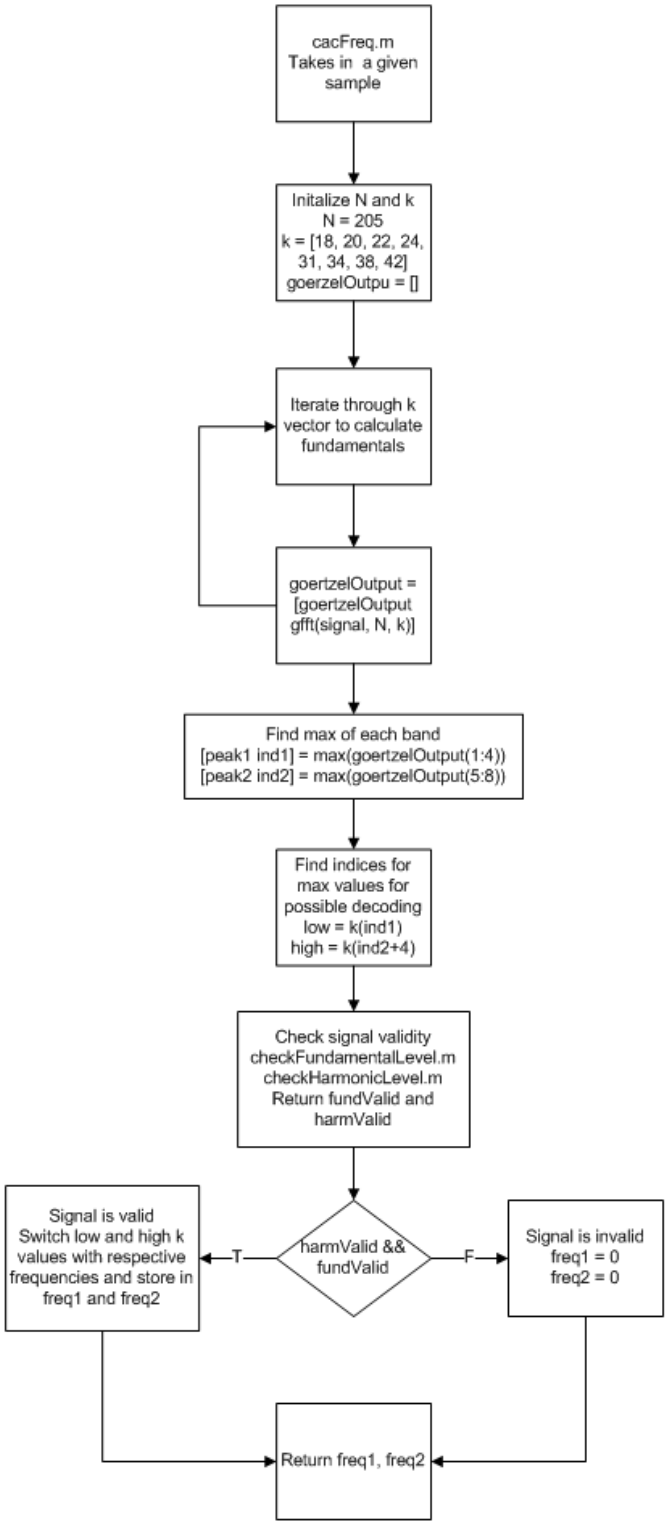
The basic process for the algorithm is as follows. The tt_decode.m program takes in a signal, and iterates through the signal in sample sizes of 200 samples. Each sample is then processed through 16 Goertzel FFTs to measure eight fundamental frequencies and eight second harmonic frequencies. These frequencies are then measured to see if a valid dial tone is present in the sample. As the program processes the data, it checks the previous sample and the current sample to see if changes occur. Random changes of silences or tones are ignored. The program will only add the detected tone if an actual break of silence is found and the tone has duration that falls within specifications. After the program has finished iterating through the signal, the resulting found digits are converted to a phone number.

2. DTMF Decoder Block Diagram

2.1. Overall Program Diagram



2.2. Frequency Analysis Block Diagram



3. Signal Processing Specifications

3.1. Analysis Window

The size of the analysis window chosen for this algorithm is 200 samples. Using this size, the duration of each sample is 25ms and the frequency resolution is 40Hz. This window allows for a proper DFT analysis with the chosen N, as any larger will cause the DFT results to be inadequate and any smaller will cause the DFT results to not have enough resolution. This segment length is enough to measure the silence durations as it includes the minimum 10ms of silence between tones and allows for easy measurement of the minimum of 50ms of silence between tones in the given samples. This value was arrived after experimentation. Using a size of 80 samples (10ms) proved to be inadequate in measuring frequencies. The next step up of 160 samples (20ms) faced the same problem. 200 samples is the highest even window size this algorithm can use with its chosen DFT N size.

3.2. DFT Size N

The DFT size for the fundamental frequencies was chosen to be N = 205, and the DFT size for the second harmonics was chosen to be N = 201. These values are suggested values for detecting the desired frequencies, and they meet the frequency specifications of accepting values with 1.5% of the tone. N = 205 is the value used for all of the fundamentals, and N = 201 is the value used for all harmonics. This value was found from iterating through different N values, and finding the values of N that fall below the 1.5%. From those results, N = 205 was chosen for this application. Using these values of N, the following k values were found and used for our Goertzel FFT algorithm:

Fundamental Frequency (N = 205)	**k value (rounded to nearest integer)**
697.0Hz	18
770.0Hz	20
852.0Hz	22
941.0Hz	24

1209.0Hz	31
1336.0Hz	34
1477.0Hz	38
1633.0Hz	42
Second Harmonic Frequency (N = 201)	**k value (rounded to nearest integer)**
1394.0Hz	35
1540.0Hz	39
1704.0Hz	43
1882.0Hz	47
2418.0Hz	61
2672.0Hz	67
2954.0Hz	74
3266.0Hz	82

3.3. Detection Threshold

In order to deal with noise and detect which signals were valid tones, two threshold mechanisms were put in place. The first threshold test (checkFundamentalLevel.m) measured the fundamental frequency power levels. The first test takes the magnitude square of the highest frequency in each band, high and low, and checks if it is greater than the sum of the magnitude square of the remaining frequencies in each band scaled by a chosen factor k (k = 7 in this algorithm). If it is greater, than the fundamental frequencies can be considered to be part of a valid tone and not just noise. The second threshold test (checkHarmonicLevel.m) involves the harmonic levels. If a second harmonic is present in the measured signal, then it is not a valid pure tone. This is in place to deal with noise and speech. This test consists of creating two measurements for the harmonics. First, it calculates the ratio of the second harmonic to the fundamental and checks if it is lower than

a specified value of .15. Second, it calculates the sum of the magnitude square of the harmonic values and checks if it falls below a specified threshold value chosen to be 150. If the harmonics pass either of these tests, the signal passes the second threshold and the signal is a pure tone. Both the fundamental threshold and the harmonic threshold must be met before a sample can be considered part of a valid tone.

3.4. Decision Logic

The decision to determine which digits are added to the output is calculated as the program iterates through the signal. As the program iterates through the signal, it checks each window sample and determines if a tone is present (calcFreq.m) in the sample and checks it against the previous tone from the previous sample. If this tone is new and it is a transition from silence or noise to a tone, then the sample is considered part of a new tone and the duration of the tone begins to be measured. If during the next samples, the tone suddenly changes to a different tone, then the new tone can be considered a new tone caused by noise since no silence was present and the program ignores it. If the next tone is silence, then the current tone may potentially have finished or the silence is caused by a sudden spurt of noise. When this occurs, the duration of silence begins to be measured. If the silence is less than the minimum length and the same tone is found once more, then the silence was due to noise and the tone continues to be measured. If the silence is greater than the minimum length of 50ms, then the previous tone has finished and the program is currently within the span of silence between tones. The tone's duration is tested to see if it falls between 40ms and 400 ms, and if it passes, it is a valid tone. The resulting digit is then added to the list of valid digits, and this process continues until the entire signal has been iterated through. The resulting string of detected digits is then converted to a phone number (formatDigit.m) that outputs at the end of the program. If for any reason, the program did not find the minimum ten digits for the phone number, the result of the decode program is 'Error.'

4. Matlab Code

4.1. tt_decode.m

```
function digits = tt_decode(x)
%tt_decode.m
%Take in a signal composed of dial tones, decode signal, and return a
%number in the format of NNN-NNN-NNNN.

%Set Sampling Frequency
Fs = 8000;

%Set Analysis Window Size and Time Division
windowSize = 200;
dt = windowSize/Fs;

%Initialize duration variables
toneDuration = 0;
silenceDuration = 0;

%Initialize Digit and Tone variables
currentDigit = 'None';
previousDigit = 'None';
previousTone = '';

%Initialize vector to hold found digits
foundDigit = [];


%Iterate Over Signal using set window size
for startIndex = 1:windowSize:length(x)-1

    %Check if enough samples are preset to allow for window analysis
    if (length(x) - startIndex >= windowSize)

        %Create windowed sample from signal
        sample = x(startIndex:startIndex+windowSize-1);

        %Calculate dual frequencies found in sample if any
        [freq1 freq2] = calcFreq(sample);

        %Check result.  If 0, there is no tone.  Otherwise, decode the
        %frequencies and find digit.
        if (freq1 == 0 && freq2 == 0)
            currentDigit = 'None';
        else
            currentDigit = lookupDigit(freq1, freq2);
        end

        %Check if the current tone found in the window matches the previous
        %tone found in the last window.
        if(strcmp(previousDigit, currentDigit))

            %Tones are the same and consistent
            if(strcmp(currentDigit, 'None'))
                %Samples are silence in a row.  Increase silence duration
                silenceDuration = silenceDuration + 2*dt;
            else
                %Samples are same tone.  Increase current tone duration
                toneDuration = toneDuration + dt;
            end

        elseif(~strcmp(previousDigit, 'None') && (strcmp(currentDigit,'None')))
            %Tones are different.  Possible that tone has finished or noise
            %has created an illusion of silence, so store the previous tone
            %in mememory, and start measuring silence duration.
            silenceDuration = 0;
            previousTone = previousDigit;

        elseif(strcmp(previousDigit, 'None'))
            %Silence is over.  Tone is detected. Reset silence duration.
            silenceDuration = 0;
```

```
            if (~strcmp(previousTone, currentDigit) && ~strcmp(previousTone, ''))
                %If current tone is different than stored tone and the
                %stored tone exists, this current tone is a wrong
                %measuremnt due to noise.  Ignore this sample.
                currentDigit = previousTone;
                toneDuration = toneDuration + dt;
            end

        end

        %Check Silence Duration
        if(silenceDuration >= .05)

            %Silence duration is long enough to be a pause between tones.
            %Previous tone is potentially a valid digit
            if(toneDuration >= 0.04 && toneDuration <= .40001)
                %Check if tone duration of previous tone is valid.  If so,
                %then a valid dial tone is added to the vector of tones.
                %Reset duration and tone variables.
                foundDigit = [foundDigit previousTone];
                previousTone = '';
                toneDuration = 0;
                silenceDuration = 0;
            end

        end
    end

    %Store current digit tone as previous digit tone
    previousDigit = currentDigit;

end

%Take resulting found digits and format output.
%Will return error if not enough digits are found.
digits = formatDigit(foundDigit);

end
```

4.2. calcFreq.m

```
function [freq1, freq2] = calcFreq(signal)
%calcFreq.m
%Calculate Frequencies in the given signal, and determine which frequencies
%for dial tones are valid and present if any.

%Initialze N and k
N = 205;
k = [18, 20, 22, 24, 31, 34, 38, 42];
goertzelOutput = [];

%Iterate through different k values, and determine Goertzel DFT outputs at
%each k.
for j = k
    goertzelOutput = [goertzelOutput gfft(signal, N, j)];
end

%Calculate peaks from low and high bands
[peak1 ind1] = max(goertzelOutput(1:4));
[peak2 ind2] = max(goertzelOutput(5:8));

%Calculate low and high indices
low = k(ind1);
high = k(ind2+4);

%Calculate the validity of the Fundamental Frequency levels and the Harmonic
%Frequency levels for the given signal at the maximum frequencies.
fundValid = checkFundamentalLevel(peak1, peak2, goertzelOutput);
harmValid = checkHarmonicLevel(signal, goertzelOutput, ind1, ind2+4);
```

```
%Check valid variables
if (harmValid && fundValid)
    %If signal is valid in fundamental frequency strength and has
    %negligible harmonics, it is a valid signal, and its low band and high
    %band frequencies are calculated.
    switch low
        case 18
            freq1 = 697;
        case 20
            freq1 = 770;
        case 22
            freq1 = 852;
        case 24
            freq1 = 941;
    end

    switch high
        case 31
            freq2 = 1209;
        case 34
            freq2 = 1336;
        case 38
            freq2 = 1477;
        case 42
            freq2 = 1633;
    end
else
    %Signal is not valid. So, return 0.
    freq1 = 0;
    freq2 = 0;
end

end
```

4.3. checkFundamentalLevel.m

```
function valid = checkFundamentalLevel(peak1, peak2, fundamentalDFT)
%checkFundamentalLevel.m
%Checks the power lever of the fundamental frequencies in the signal.
%Returns 1, if this signal is a valid tone.  Returns 0, otherwise.

%K is a threshold. The magnitude square of the peaks must be K times
%greater than the sum of the magnitude squares of the remaining frequencies
%in the band for this signal to have enough power to be considered a tone
%and not noise.
k = 7;

valid = ( ( abs(peak1)^2 >= k*(sum((abs(fundamentalDFT(1:4)).^2)) - abs(peak1)^2) ) ...
        && ( ( abs(peak2)^2 >= k*(sum((abs(fundamentalDFT(5:8)).^2)) - abs(peak2)^2) ) ) );

end
```

4.4. checkHarmonicLevel.m

```
function valid = checkHarmonicLevel(signal, fundamentalDFT, ind1, ind2)
%checkHarmonicLevel.m
%Checks the Harmonics present in the signal.  Returns 1, if harmonics are
%negligible.  Returns 0, if harmonics are too high indicating that this is
%not a pure tone.

%Initializes N and k
N = 201;
k = [35, 39, 43, 47, 61, 67, 74, 82];
harmonics = [];

%Sets a threshold value.  If the sum of the magnitude square of harmonics
%is less than this value, then the harmonics are negligible.
thresh = 150;
```

```
%Iterate through k values, and calculate DFT for each harmonic frequency
for j = k
    harmonics = [harmonics gfft(signal, N, j)];
end

%Calculate the ratio of harmonic to fundamental for the max values.  If the
%ratio is too high, then this signal is not a pure tone.
ratio = mean(abs(harmonics([ind1 ind2])) ./ abs(fundamentalDFT([ind1 ind2])));

%Valid if the harmonics meet one of these condiditons.
valid = (sum(harmonics.^2) < thresh) || (ratio < .15);

end
```

4.5. lookupDigit.m

```
function digit = lookupDigit(f1, f2)
%lookupDigit.m
%Takes frequencies and returns the corresponding digit for those two
%frequencies.

%Initialize Digit Lookup Table
digitTable = ['1', '2', '3', 'A';
              '4', '5', '6', 'B';
              '7', '8', '9', 'C';
              '*', '0', '#', 'D'];

%Using given frequencies, determine row and column of digit table
switch f1
    case 697
        i = 1;
    case 770
        i = 2;
    case 852
        i = 3;
    case 941
        i = 4;
end

switch f2
    case 1209
        j = 1;
    case 1336
        j = 2;
    case 1477
        j = 3;
    case 1633
        j = 4;
end

%Find the indicated digit
digit = digitTable(i,j);

end
```

4.6. formatDigit.m

```
function digits = formatDigit(digitVector)
%formatDigit.m
%Takes a string of digits and formats the input into a phone number

%If the string of digits is too short, then return 'Error' indicating that
%a valid phone number is not returned.
if length(digitVector) < 10
    digits = 'Error';
else
    digits = [digitVector(1:3) '-' digitVector(4:6) '-' digitVector(7:10)];

end
```