DTMF Digit Detection Algorithm

1 Project Approach

This project is almost identical to the technical challenges we faced in ECE 301, discrete time signals and systems. The challenge, as discussed in person with Dr. Lu, was to adapt our code from ECE 301 to improve upon our existing Dual Tone Multiple Frequency (DTMF) Detection model. In particular, Dr. Lu took issue with our approach of manually segmenting our input signals, therefore effectively eliminating the noise and timing challenges imposed by the problem statement. Our solution to this issue was to create a moving window DTMF detection system which replied upon an ecosystem of functions to quickly and generally solve smaller technical challenges.

2 Moving Window Process

Within our function ecosystem, we identified three tasks which needed to be accomplished: finding frequency peaks, identifying key pressed from peaks and managing a moving window system to manage detection and to avoid duplicate true positives.

2.1 TwinPeaks.m

function fPair = TwinPeaks(sample, freq)

The twin peaks function is the heart of the DTMF detection operation. It performs a short fast-Fourier transform on the current window and forms an amplitude spectrum. This spectrum will illustrate noise below 600 Hz and will also show the peaks of the dual-tone frequencies between 600 Hz and 2000 Hz if they are present. Instead of applying a high pass filter with a cutoff of 600 Hz, thus eliminating noise, our algorithm saves valuable computing time by only analyzing the frequency range which interests us: 600 to 2000 Hz.

Using the spectrum, we find the two greatest peaks in our DTMF range and return them as a vector. By creating this function we can generalize our approach so that it can analyze any window width for any sampling rate of any recording.

2.2 IdentifyKey.m

function [match, kVal] = IdentifyKey(freqPair, tol)

The next function used in our software was a system to take peak frequencies and check them against a table of DTMF pairs. We designed this function with a tolerance input which can be calibrated accordingly. If the input frequency pairs were within ± tol of a valid frequency pair then the function returned match = True as well as the kVal variable which stored the key pressed. If no valid pair was detected then it returned match = False. The match variable enabled process flow control outside of our lower-level functions.

2.3 DTMF.m

function phone_number = DTMF (sample, time, frequency, window_width, ext, plt)

Combining the efforts of its sub-functions, the DTMF function manages the logic of the algorithm. In particular, the moving window approach demands the detection of digits and the

avoidance of duplicates when the window steps to the next section. Our system bypasses this issue by relying on the above functions instead of having all of the code under one MATLAB file. This enabled us to easily repeat functions and develop a higher level logic.

The window begins looking at the first *window_width* number of values in the sample and automatically assigns a step size of half the *window_width*, ensuring that no digits can be missed due to the window overlap. The window steps through the sample and waits until it receives a *match* from *IdentifyKey.m*, see Figure 1 part A. At that point, it appends the matched key to the *phone_number* variable and steps through subsequent sample windows to avoid duplicates, Figure 1 part B. When it samples a window without valid dual tone frequencies, it exits the loop and iterates through the sample until it collides into another key press, Figure 1 part C. At this point it repeats the process until it runs out of data to sample.

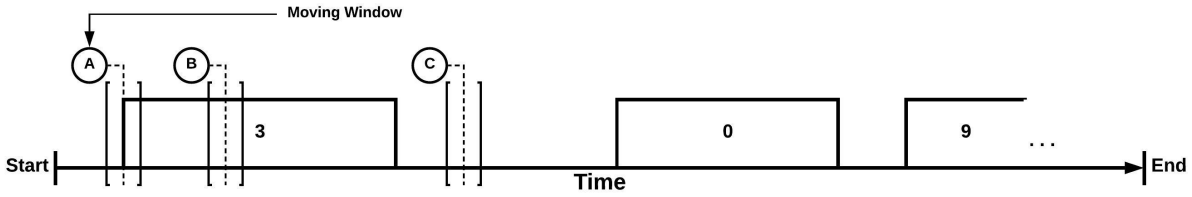


Figure 1: Moving window detection stages.

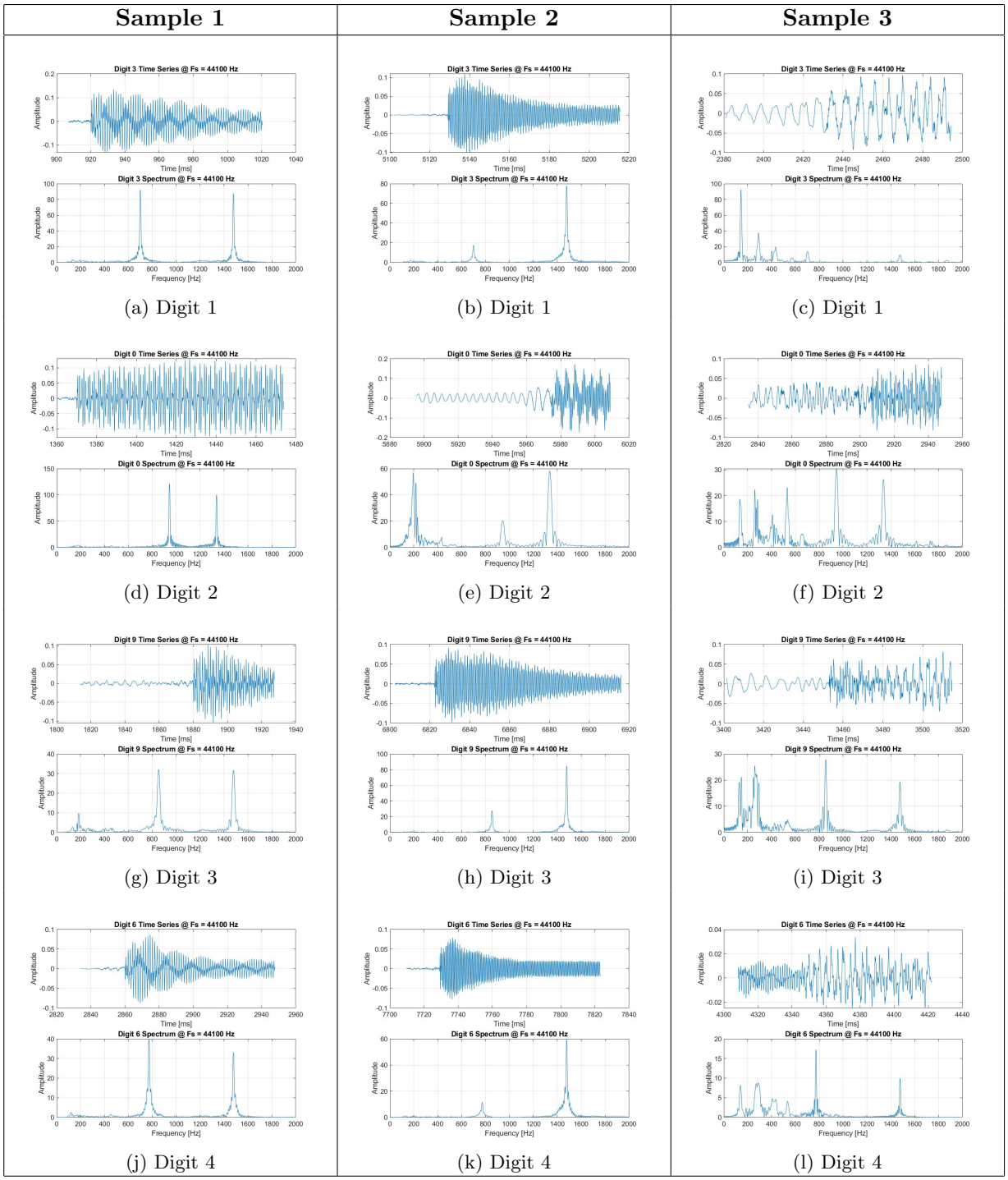
Expressed more explicitly, Figure 3 in the appendix illustrates how the process flow detects digits and avoids duplicate detection. Additionally, it shows how the DTFM function is used in the bottom left-hand box. Here we can see that each sample can be easily loaded and the phone number can be returned.

3 Algorithm Results

Inspecting the function definition under section 2.3, the function takes in an argument for plotting, *plt*. When this argument is set to *1*, plots of each matched digit will automatically be created and saved under the Plots directory. If settings are changed and executed again then these plots will be overwritten. In addition to the *plt* input, DTMF accepts a string extension (*ext*) for each plot saved under its call. This enables the user to call the function under different conditions or for different samples and generate plots which are labeled specifically for that call.

Using the two inputs described above, we generated three sets of 10 plots, one plot per DTMF digit identified and one set per sample. some of these plots have been displayed on the subsequent page.

We can see that the noise becomes more prominent in the sub 600 Hz frequency range with samples 2 and 3, compare Figures 2 (d), (e) and (f). Because *TwinPeaks.m* (See 2.1) only checks for peaks between 600 and 2000 Hz, this noise has never become an issue. Notice in all samples that the end of the time series plot if truncated short of the end of the sound recording. Consequently, although there is sufficient noise in this range to cause a problem for samples 2 and 3, our process is unaffected by noise within this frequency range. If, on the other hand, there was noise or distortion between 600 and 2000 Hz, then our algorithm would be insufficiently robust to distinguish between noise and DTMF key presses.



(a) Digit 1 (b) Digit 1 (c) Digit 1

(d) Digit 2 (e) Digit 2 (f) Digit 2

(g) Digit 3 (h) Digit 3 (i) Digit 3

(j) Digit 4 (k) Digit 4 (l) Digit 4

Figure 2: First four digit readings per sample.

As we mentioned earlier, our solution can scan, plot, and return DTMF key presses quicker than previous solutions. Using MATLABs *tic toc* commands we timed the performance of our DTMF calls. After each iteration it became faster due to MATLABs dynamic memory allocation but it consistently performed key detection in under half a second, see the appendix for Figure 4.

4 Appendix

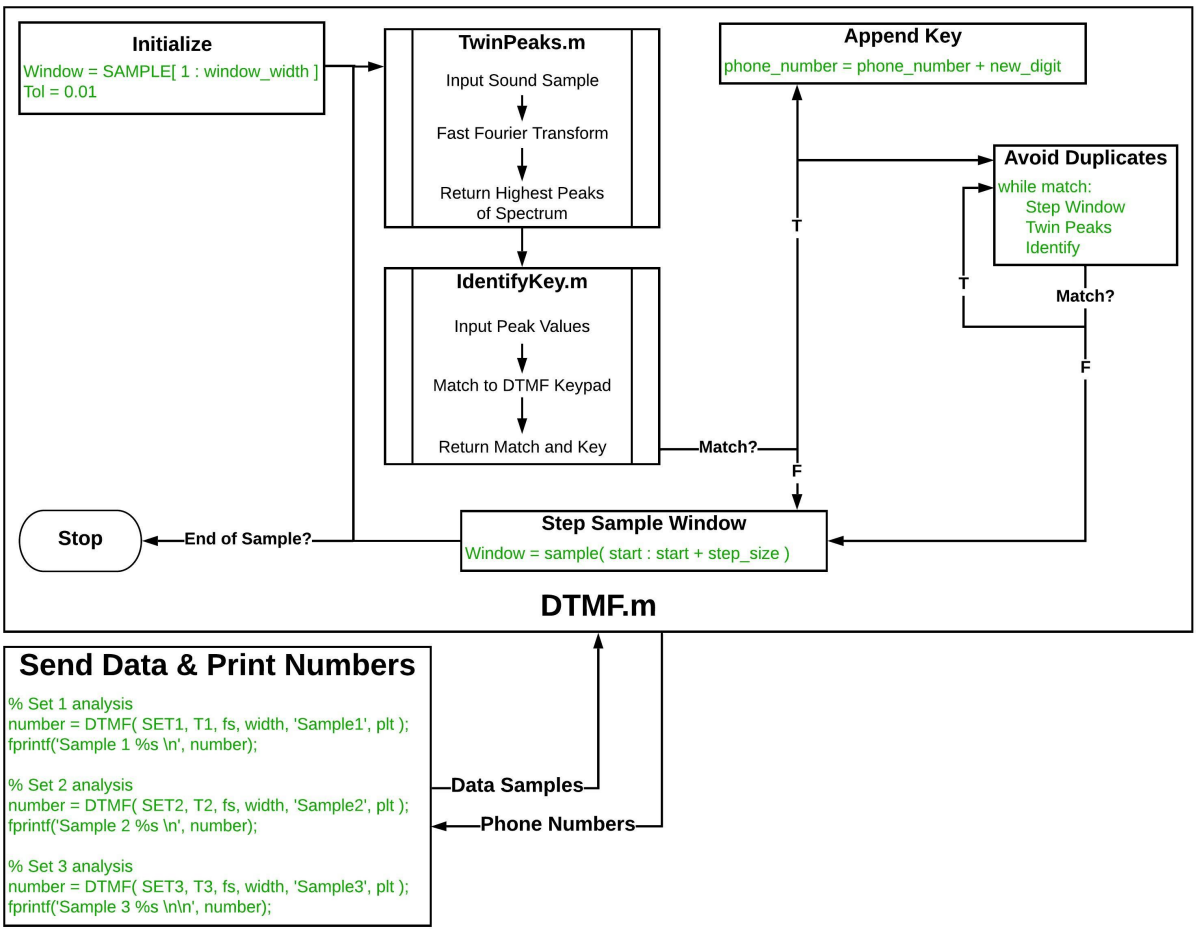


Figure 3: Broad program ecosystem.

```
Sample 1 Phone Number: 3096772734
Elapsed time is 0.251216 seconds.
Sample 2 Phone Number: 3096772734
Elapsed time is 0.139618 seconds.
Sample 3 Phone Number: 3096772734
Elapsed time is 0.081017 seconds.
```

Figure 4: DTMF speed test.