DIGITAL SIGNAL PROCESSING

WORD RECOGNITION FOR CAPTCHA SECURITY TEST

Student Name: Asmaa Adel Alazmi *Student ID#*: 2201122708

Email: s2201122708@eng.ku.edu.kw *Date*: April 3, 2024

ABSTRACT

This project utilizes the functionalities of bandpass filters and the frequency domain energy formula for a given audio signal to differentiate between words. The computation in this project is done in the frequency domain after converting the signal from the time domain using the Fast Fourier Transform (FFT) on the signal. The algorithm is then used in a simple Captcha security model on a web page.

1. INTRODUCTION

Digital signal processing (DSP) methods are a necessity for all modern technologies. When utilizing the differences in the frequency spectrum between words more specifically letters, it's possible to differentiate between said words using a system that can receive audio signals. The aim of this project is to integrate the qualities of basic DSP concepts and methods into the realm of cyber security. Using a word recognition system can easily distinguish between a bot or a human if applied in a Captcha test before entering a webpage or requesting a response from a server. For the following project the word recognition system was applied using two words: 'yes' and 'no'. By keeping the Captcha test straightforward with short, easy-to-understand words, it ensures that nearly everyone can successfully complete it, even without extensive English language knowledge.

2. GENERAL INSTRUCTIONS

This section covers the theoretical part of the project.

2.1. Digital Signal Processing (DSP)

The term signal is generally applied to something that conveys information. DSP is a field of study that involves manipulating digital signals to analyze or modify those signals. The main role of this field is to transform real-world signals (continuous signals) into a digital computable format (finite signal), that is efficient and easily implemented through the wide spectrum of modern technologies.

2.1.1. Digital Audio Signals

Digital audio signals are represented by an array of discrete sequences of data points captured over a period of time. What differentiates digital audio signals from analog audio signals is their discontinuity. Unlike analog, which flows continuously, digital signals are finite representations. Using such finite signals allows machines to communicate and compute those signals. Handling infinite continuous signals would demand limitless computational power and time, which is beyond current capabilities.

2.1.2. Fast Fourier Transform (FFT)

The fast Fourier transform is an algorithm to compute the discrete Fourier transform. It transforms a sequence of discrete data in the time domain into a sequence of imaginary and real numbers (complex numbers) in the frequency domain. The FFT outstands as the fastest method to achieve this transformation dominating its predecessor the discrete Fourier transform (DFT) by having a lower computational complex time and reducing the computational cost and saving resources. And this achievable due to decomposing the DFT matrix into a product of sparse factors. FFT yields a symmetric matrix as a result, simplifying programming efforts by requiring consideration of only the unique first half of the data, thereby reducing time complexity for developers and analysists. The mathematical representation of the FFT is:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i\frac{2\pi}{N}kn}$$

Such that:

- **X[k]:** the output FFT matrix in the frequency domain.
- **x[n]:** the input discrete audio signal.
- **N:** the number of discrete samples.

Using the FFT algorithm also offers the advantage of mapping each magnitude with the frequency it corresponds to, as will be further discussed in the code section.

2.1.3. Energy Formula

The energy of a signal can be computed using the energy formula. In this project, we compute the signal's energy after transforming it into the frequency domain using FFT. The energy in the frequency domain can be determined using a straightforward mathematical expression.

$$E = \frac{1}{N}\sum_{k=0}^{N-1}|X[k]|^2$$

2.2. Low-pass filter

Low-pass filters serve as the fundamental building block for various filter designs. Functioning by permitting signals with frequencies below a specified cutoff frequency to pass through while attenuating frequencies beyond this threshold. In an ideal scenario, a low-pass filter would seamlessly transition between passing and attenuating frequencies, achieving a sharp transition. However, in practical application, this transition from passband to stopband cannot be instantaneous, introducing alternative approaches and considerations in filter design.

2.2.1. Low-pass elliptic filter

The Elliptic filter, also recognized as the Cauer filter. It's a signal processing filter with equalized ripple behavior in both passband and stop-band filters. This type of filter stands out as no other filter of equal order can have a faster transition in gain between the passband and the stopband. That is why it is the ideal solution for this project as it offers a fast transition between the passband and the stopband, almost emulating an idle filter. The mathematical representation of the magnitude response of a low-pass Elliptic filter is as follows [1] [5] [6]:

$$|H(\Omega)|^2 = \frac{1}{1+\varepsilon^2 u_N^2(\Omega)}$$

Such that:

- $\boldsymbol{\Omega}$: is the normalized frequency (rad),
- $\boldsymbol{u_N(\Omega)}$**:** is a Jacobian elliptic function.
- $\boldsymbol{\varepsilon}$**:** the epsilon affects the ripple characteristics of the filter in the frequency domain. As it increases the ripples in the frequency response increases.

The Jacobian elliptic function results in the passband and the stopband having an equal ripple error. By having an equal ripple error, it allows the transition from the passband to the stopband to be as narrow as possible for a given filter order. A discussion on the Jacobian elliptic function, even on a superficial level, is beyond the scope of this report [7].

A visual representation of the low pass elliptic filter in the frequency domain (Figure 1):

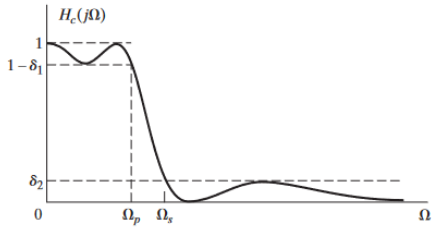

Figure 1 - Elliptic low pass filter

Where:

- $\boldsymbol{\Omega_s}$**:** specifies the stopband edge frequency.
- $\boldsymbol{\Omega_p}$**:** specifies the passband edge frequency.
- $\boldsymbol{\delta_1}$: Passband ripple or deviation.
- $\boldsymbol{\delta_2}$: Stopband ripple or deviation.

2.2.2. Bandpass elliptic filter

Bandpass filters are a type of filter that companies both the functionalities of a low pass filter and a high pass filter. It allows signals with frequencies lower than a certain high-cut frequency (Ω_{c2}), and higher than a low-cut frequency (Ω_{c1}) to pass. While it attenuates any frequencies outside that range (in an ideal case). To design a bandpass filter the formula of a low-pass filter and a high-pass filter are needed.

We can represent the frequency response of the filter as:

$$H_{LPF}(\Omega\mathrm{j}) = |\mathrm{H}(\Omega)|e^{-\mathrm{j}\phi(\Omega)}$$

Where, $\phi(\Omega)$ is the phase response of the low-pass filter. Now, to convert this to a high pass filter using phase shift method, a phase shift of π is applied:

$$H_{HPF}(\Omega\mathrm{j}) = |\mathrm{H}(\Omega)|\, e^{-\mathrm{j}\phi_{shifted}(\Omega)}$$

After that combining both formulas we can have a bandpass filter:

$$H_{BPF}(\Omega\mathrm{j}) = H_{LPF}(\Omega_{c1}\mathrm{j}) + H_{HPF}(\Omega_{c2}\mathrm{j})$$

However, it's worth noting that the phase response of an elliptic filter is generally more complex compared to other filter types due to its ripple characteristics. Therefore, the phase response may not have a simple analytical expression and might need to be computed numerically or approximated using specialized methods. That is why it's best implemented using code as will be discussed further on.

2.2.3. Choosing the Filter's Order

The order of the filter was established using Matlab, as Matlab offers the ability to generate the minimum filter order required with ease and precision. The function (`ellipord`) returns the order of an elliptic filter, it requires a set of specs consisting of the sampling frequency, Nyquist, passband, and stopband frequencies. It also

requires the passband and the stopband ripples in (dB). After running this function with the wanted specs the code generated that the minimum order of 4 for the bandpass filter. The code was as following:

```
Fs = 44100; % Sampling frequency (Hz)
F_Nyquist = Fs / 2; % Nyquist frequency

% Passband frequencies normalized to Nyquist frequency
Wp = [200 300] / F_Nyquist;

% stopband frequencies normalized to Nyquist frequency
Ws = [150 350] / F_Nyquist;

Rp = 0.5; % passband ripple (dB)
Rs = 40; % stopband ripple (dB)
[n, ~] = ellipord(Wp, Ws, Rp, Rs); % n = order
```

The Nyquist frequency (`F_Nyquist`) represents half of the sampling rate (`Fs`). It's the maximum frequency that can be represented in the digital signal without aliasing. Calculating the Nyquist frequency is crucial for determining the frequency range of the filter. The frequencies (`Wp` and `Ws`) which represent the passband and the stopband respectively are normalized by dividing them by the Nyquist frequency. Normalization ensures that the cutoff frequencies are expressed relative to the Nyquist frequency. The (`Rp=0.5`) denotes the passband ripple flatness, opting for a lower value flattens the frequency response to minimize distortion. However, this can introduce increased computational needs. By setting it to 0.5, we strike a balance, avoiding extreme trade-offs. Lastly, the parameter (`Rs=40`) represents the stopband attenuation level. Lowering the stopband attenuation requirement (`Rs`) to 40 dB means that you are allowing more signal from the stopband to leak into the passband. This relaxation of the stopband attenuation requirement typically leads to a less stringent filter design, potentially resulting in a lower filter order, which is cost efficient. Further justification for the specs will be discussed.

2.2.4. Input Audio Signal

The project's code processes .WAV input files, while the website implementation handles live recorded audio. For optimal performance, it's recommended to use a microphone equipped with a built-in noise cancellation system. This feature enhances signal clarity and minimizes the chance of errors, particularly those resembling wind noise. Wind noise tends to manifest at lower frequencies, potentially affecting the final result. The system is designed to accommodate both male and female voice inputs and yields accurate outcomes when the conditions are met.

3. WORD RECOGNITION

Word recognition depends on the analysis of signals in the frequency domain. By sampling the frequency components of spoken words, the system deciphers and transcribes them into text. This approach harnesses basic DSP techniques to enhance accuracy and efficiency.

3.1.1. Differentiating Between Words

Each letter in the English alphabet corresponds to a distinct frequency range, contributing to its unique sound signature when spoken. Using this concept enables the differentiation between letters and short words. As discussed previously, the energy of a signal can be calculated in the frequency domain. In theory, the word ‘no’ for example, operates within lower frequency ranges, suggesting it possesses higher energy within this frequency range.

3.1.2. Frequency Range Analysis

In this project, extensive testing was conducted on multiple input samples for the words 'yes' and 'no'. The objective was to identify the frequency range with the highest energy for each word. This range of frequencies would then be utilized for word recognition purposes. Using a Python code, the input samples were analyzed with frequency ranges with step of 100 Hz. The unmodified spectrum ranged from 0 Hz to 8 kHz for each signal, as most letters operate within frequencies lower than 8 kHz. The code calculates the energy for each frequency range and determines the range with the maximum energy. To loop through the frequency range with steps of 100 an elliptic filter of order 4 was used. The (ω_{c1}) corresponds to the start frequency of the range and the (ω_{c2}) takes the value of the end frequency of the range. To ensure accuracy, the code is run on multiple data samples for each word, and the mean frequency range is derived from the final result. The code is implemented in the following function:

```
def mean_freq_range():
    mean = []
    directory='yes' #test the yes sample files

    for filename in os.listdir(directory):
        file_path = os.path.join(directory, filename)
        # Read the audio data from the WAV file
        rate, data = adjust_wav_file(file_path)
        energies = []

        for f in range(200, 8000, 100):
            filtered_data = BPF(data, f-99,f,rate,order=4)
            mag, freq = compute_FFT(filtered_data)
            e = calculate_energy(mag)
            energies.append(e)
        mean.append((energies.index(max(energies))+2)*100)

     w_c=sum(mean)/len(os.listdir(directory)) #center freq
     w_c1 =  w_c - 50 # low cutoff frequency
     w_c2 = w + 50 # high cutoff frequency
```

Further clarification of the code and its functionality are discussed in the Design & Implementation section.

3.1.3. Results & Evaluation

The algorithm's outcomes revealed that the word 'no' exhibits the highest energy within the frequency range of

200Hz to 300Hz. This suggests that for implementing a bandpass filter, the cutoff frequencies (ω_{c1}) and (ω_{c2}) would be set at 200Hz and 300Hz (Figure 2), respectively. For the word 'yes', the energy peak occurred within the range of 450Hz to 550Hz (Figure 3). Alas, it's important to note that the letter 's' in the word also operates at much higher frequencies, above 6.5kHz (Figure 4), which necessitates consideration when computing the overall energy. Finally, to indicate which word was said, a comparison between the energies should suffice. Two bandpass filers should be applied, each one should carry the frequency boundaries of a single word, additionally an extra HPF for the letter “s” in “yes” is needed. Then the energy of the resulted data should be computed. The filter with the highest energy should indicate the word that was initially said. The following flow chart displays the steps to executing the word recognition algorithm in Python (Figure 5).

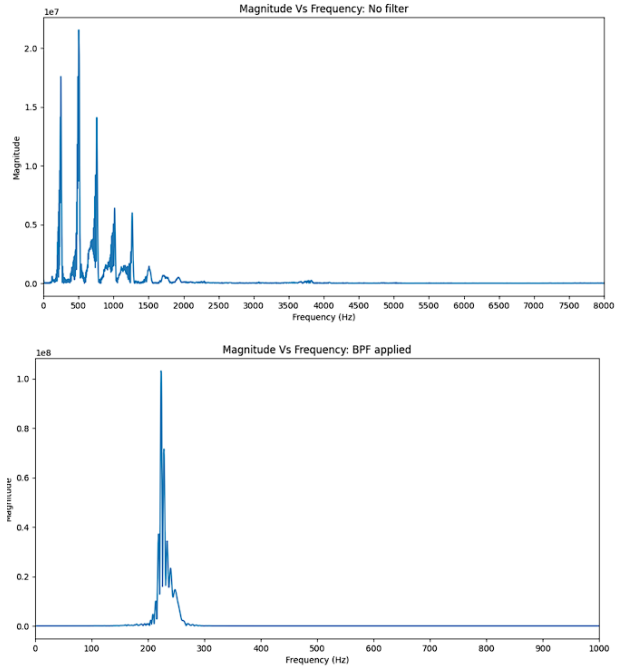

Figure 2 - Testing the word “no”

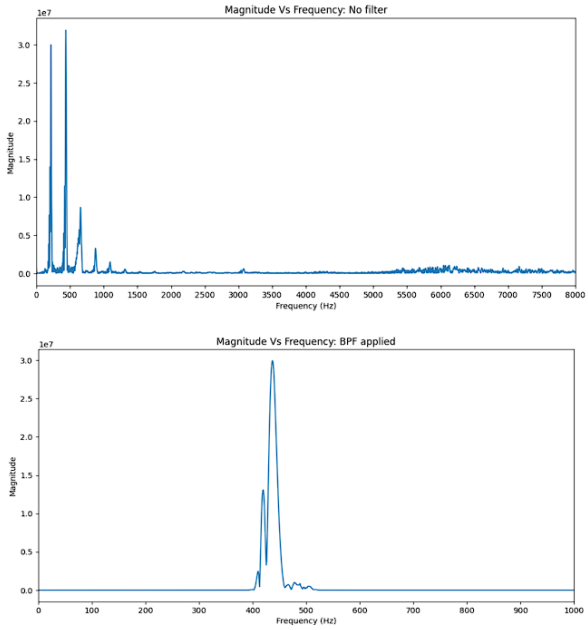

Figure 3 – Testing the word “yes”

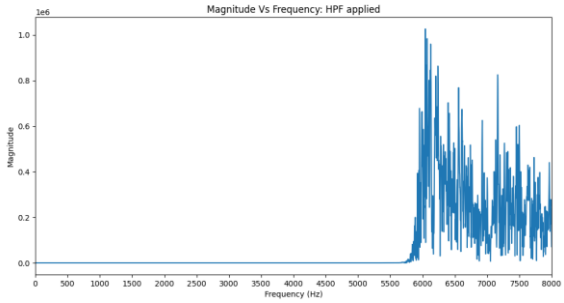

Figure 4 - HPF for the letter “s”

Figure 5 - Algorithm flow chart

4. DESIGN & IMPLEMENTATION

The implementation of this project is divided into 3 main parts. Each part will reference code snippets and explanation for the design decision made in that code. The project was code using Python and the user interface was made using the Streamlit API (Application Programming Interface) library. Also, the Python libraries NumPy and SciPy were used for computational and data manipulation purposes.

4.1.1. Input Data Format

After receiving audio data from a user, it's crucial to ensure that the data meets specific criteria to enhance its analysis using Python. The audio format from a mic is usually a stereo audio, which means that the audio runs on two channels: left and right. The audio of the file must be a monophonic sound (mono), meaning that the audio signal must be represented as a single channel. Additionally, mono audio files typically require less storage space and bandwidth compared to stereo files. By converting from stereo to mono audio the process of analyzing audio signals is extremely simplified. The audio sample rate, representing the number of samples captured per second, was chosen in adherence to the national standard for audio format, set at 41 kHz per second. Ensuring each sample in the audio file maintains a 16-bit encoding (2 bytes per sample) through adjustment enhances the accuracy of results. While higher bit rates generally correlate with improved signal quality, this improvement comes at the expense of increased computational demand. Finally, after adjusting the file the function shall return the resulted rate which should be 41kHz and the new NumPy data array.

```
FORMAT = pyaudio.paInt16  # format of sampling 16 bit int
CHANNELS = 1
RATE = 44100  # number of sample in 1 second sampling
BYTES = 1024  # bytes per second

def adjust_wav_file(file):
    WAVE_INPUT_FILENAME = file
    # Read the audio data from the WAV file
    rate, data = wav.read(WAVE_INPUT_FILENAME)

    # Convert to mono
    if len(data.shape) > 1:
        data = np.mean(data, axis=1)

    # Resample
    if rate != RATE:
        data = resample(data, int(len(data) * RATE / rate))

    # Adjust byte rate if necessary
    bytes_per_sample = data.dtype.itemsize

    if bytes_per_sample != 2:
        # Convert data to 16-bit int (if not already)
        data=(data/np.max(np.abs(data))*(2**15-
       1)).astype(np.int16)

    # Adjust number of channels if necessary
    if data.ndim > CHANNELS:
        data = data[:, 0]  # Keep only one channel
    return rate, data
```

4.1.2. Applying the Digital Elliptic Filters

The newly formatted data is now prepared for filtering using the discussed filters. The SciPy library provides the ellip() function, which serves as a digital finite elliptic filter. However, this imported filter exclusively accepts discrete data values as input and produces discrete values as output. This limitation can impact time complexity and computational resources, as multiple redundant domain transformations are required in the project's algorithm, as illustrated in Figure 4. The filter takes discrete values x[n], transforms them to the frequency domain X($e^{j\omega}$), constructs the frequency response filter H($e^{j\omega}$), computes the result Y($e^{j\omega}$), and then converts it back to the time domain, yielding the value y[n]. The following code shows the implementation of the filter:

```
def bandpass_filter(data, lowcut, highcut, fs, order):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = ellip(4, 0.5, 60, [low, high], btype='band')
    y = filtfilt(b, a, data)
    return y
```

```
def highpass_filter(data, cutoff_freq, fs):
    nyquist = 0.5 * fs
    high = cutoff_freq / nyquist
    b, a = ellip(8, 0.5, 60, high, btype='highpass')
    y = filtfilt(b, a, data)
    return y
```

As discussed previously in the Matlab filter order code (2.2.3), the same specs were used in this Python code and both filter orders were retrieved from that code's result. But the stopband ripple was modified to be 60dB to reduce the signal leakage from the passband. For this Python code, the designed filter coefficients (`b`) numerator and (`a`) denominator are applied to the input data (`data`) using the (`filtfilt`) function. This function performs zero-phase filtering, which filters the data forward and then backward to remove phase distortion. It ensures that the filtered signal aligns well with the original signal in terms of phase. In the ellip function, the first parameter (4) in the BPF and (8) in the HPF signifies the filter order.

4.1.3. Applying the FFT Algorithm

Following the previous step, the function results in y[n]. The next step is to take the filtered data y[n] and pass it through the FFT algorithm such that the array of data is converted to the frequency domain to enable computing the energy and plotting the magnitude vs frequency spectrum. The FFT algorithm can be implemented using the NumPy (np) library as follows:

```
def compute_FFT(data):
    fft = np.fft.fft(data)
    mag = np.absolute(fft)
    mag = mag[0:int(len(mag)/2)]
    # Frequencies corresponding to FFT result
    frequencies = np.fft.fftfreq(len(data), 1/RATE)
    return mag, frequencies
```

The FFT yields a symmetric array of data, allowing to compute half the magnitude array only, thus reducing time complexity. To map frequencies accurately, we utilize the (`fftfreq`) function, which generates an array of frequencies aligned (index-wise) with their corresponding magnitudes. This enables plotting the magnitude vs frequency.

4.1.4. Calculating the Energy

The next step after applying each filter and retrieving the filtered data in the frequency domain is to calculate the energy of each set of Y[k] produced by each filter. To compute the energy in the frequency spectrum the following algorithm is implemented using the summation function from the NumPy library (np.sum()):

```
def calculate_energy(magnitude):
    return np.sum(magnitude**2)/len(magnitude)
```

4.1.5. Comparing the Results

Following energy computation, the energy from the 'yes' BPF is combined with the 's' HPF energy, as depicted in Figure 4. To conclude the spoken word, an energy comparison is conducted between the 'yes' and 'no' energies. For enhanced accuracy, a minimum energy threshold for frequencies above 6.5kHz is employed to detect the letter 's'. Failure to meet this threshold indicates signal noise (mostly wind noise), resulting in a reversal of the word recognition outcome. The threshold was determined by conducting multiple sample tests for the word 'yes' and selecting the minimum energy value obtained from the HPF. The following code snippet represents the implementation:

```
min_threshold = 13116058260.837112 #threshold for 's'
mute_threshold = 125010299572.6434 #if the user said nothing
if energy_no < mute_threshold:
      return 'MUTE'
if energy_no > energy_yes:
     if energy_6500 > min_threshold:
           return 'YES'
     else:
                return 'NO'
elif energy_no < energy_yes:
     if energy_6500 < min_threshold:
           return 'NO'
     else:
           return 'YES'
```

4.1.6. Captcha Test

Utilizing the same algorithm and implementation, a straightforward Captcha test can be integrated into a website. The website was programmed using Python, and with the Streamlit library, audio input can be directly captured from the user's microphone. The Captcha test starts by presenting a random image containing the word 'yes' or 'no'. Subsequently, the user is instructed to say the word displayed in the image. Using the designed algorithm, if the user accurately says the correct option, the website unlocks, granting continued browsing access. Conversely, if the Captcha test outcome is incorrect, the website locks, prompting the user to retry again (Figure 6, 7).

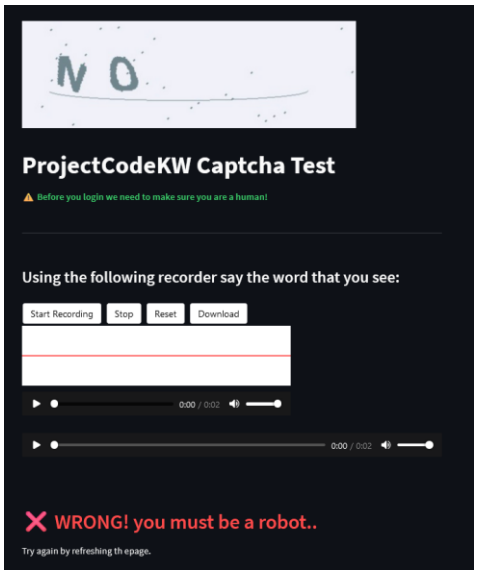

Figure 6 - Failing the Captcha test

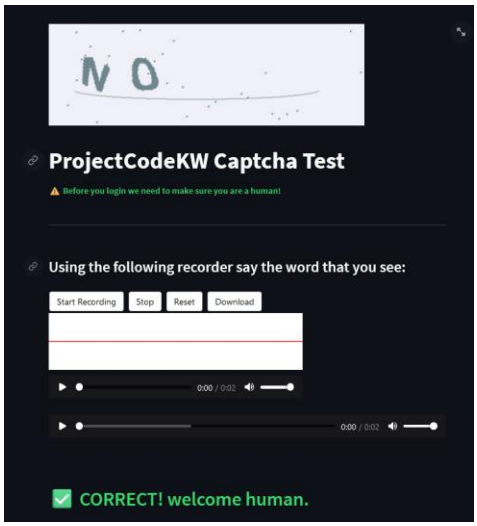

Figure 7 - Succeeding the Captcha test

5. SOURCES

- The demo Captcha website can be accessed here.
- The source code can be accessed here.

6. CONCLUSION

The project successfully met its initial objective by implementing an algorithm for word recognition, employing two elliptic filters to filter the input data for the FFT algorithm depending on a certain range of frequencies for each word. This resulted in a frequency domain array, enabling the plotting process. Through computing the energy of each word from the resulted FFT, the word recognition app was able to accurately detect the user's voice input for the Captcha test and depending on it either granting them access to the website or blocking them out. In summary, the utilization of digital filtering techniques and fundamental DSP concepts, as demonstrated in this project, can enhance nowadays security tests and authentication protocols. These concepts serve as the foundation for modern-day technologies.

7. ACKNOWLEDGMENTS

I wish to express my gratitude to Dr. Abdullah Al-Qallaf and Eng. Yousef Mullayousef for their continuous support and guidance during the duration of this project.

8. REFERENCES

[1] Oppenheim, A., & Schafer, R. *Discrete-time signal processing third edition*. Pearson. 2010.

[2] Elliptic filter. In Wikipedia. Retrieved April 27, 2024, from https://en.wikipedia.org/wiki/Elliptic_filter

[3] Orfanidis, S. (2006). *Notes on theory and applications of DSP* [PDF document]. Retrieved from https://www.ece.rutgers.edu/~orfanidi/ece521/notes.pdf

[4] Orchard, H., & Willson, A. (1997*). Elliptic Functions for Filter Design*. IEEE Transactions on Circuits and Systems—I: Fundamental Theory and Applications, 44(1), 33-43.

[5] Barry Van Veen. (2012, Dec 31). *Continuous-Time Chebyshev and Elliptic Filters* [Video]. YouTube. https://www.youtube.com/watch?v=lRQ6-Va-7EQ&ab_channel=BarryVanVeen

[6] Technologies Discussion. (2022, Dec 1). *Compare Between Bessel Vs Butterworth Vs Chebyshev Vs Elliptic Filter Designs* [Video]. YouTube. https://www.youtube.com/watch?v=_LB-HRMcGzc&ab_channel=TechnologiesDiscussion

[7] El-Tawil, M., Elhanbaly, A., & Abdel-Rehim, A. (2014). *Jacobi Elliptic Function Solution for Solitary Wave, Periodic Wave, and Periodic Wave Trains.* Journal of Applied Mathematics, 2014.

[8] Ahmad Dar Khalil. (2017, Nov 17). *Build a system to classify the sound to 'yes' or 'no' based on energy* [Video]. YouTube. https://www.youtube.com/watch?v=6Fs2-L93SGA&list=LL&index=4&t=827s&ab_channel=AhmadDarKhalil