FHSS Report 1

In this application, Frequency-Hopping Spread Spectrum (FHSS) is utilized to generate a transmission. This process encourages us to divide a wide frequency range into smaller sub-channels. After that hopping between these sub-channels in a pseudorandom sequence during signal transmission.This method increases its resistance to interference from outside sources.

In the context of Transmitter:

1. **GUI:** A Graphical user interface is established to manipulate input data and some parameters.
2. **Frequency Table Generation:** A table of possible frequencies is created based on the ID number of the student and the selected signal category.
3. **Input data Converter:** A method is established to convert string input binary coded input, in that way it can be transmitted.
4. **Pseudorandom Frequency Selection:** A random sequence is used in order to select frequencies from the generated table for signal modulation.
5. **Signal Modulation:** Signal is modulated using the selected frequencies.
6. **Signal Generation**: A signal is generated according to selected frequencies and hoping period..
7. **Spectrogram Visualization:** The hopping pattern is visualized through a spectrogram.

GUI

The GUI for the transmitter program can be seen in figure 1. This program allows user access for input data, signal category and sampling rate. Large green "Run" button to execute the main processes. Below these controls, a spectrogram visualization panel displays the frequency content of the signal over time. In figure 2 and figure 3 we can see how we manipulate the signal category and sampling rate selection in GUI.

The GUI for the receiver program can be seen in figure 4. This program allows user access for signal category and sampling rate. Large green "Record" button to execute the main processes. Below these controls, a spectrogram visualization panel displays the frequency content of the signal over time. Moreover, there is a toggle which changes the spectrogram between filtered signal and unfiltered signal. Decoded data and Estimated SNR are represented in the GUI of the receiver.

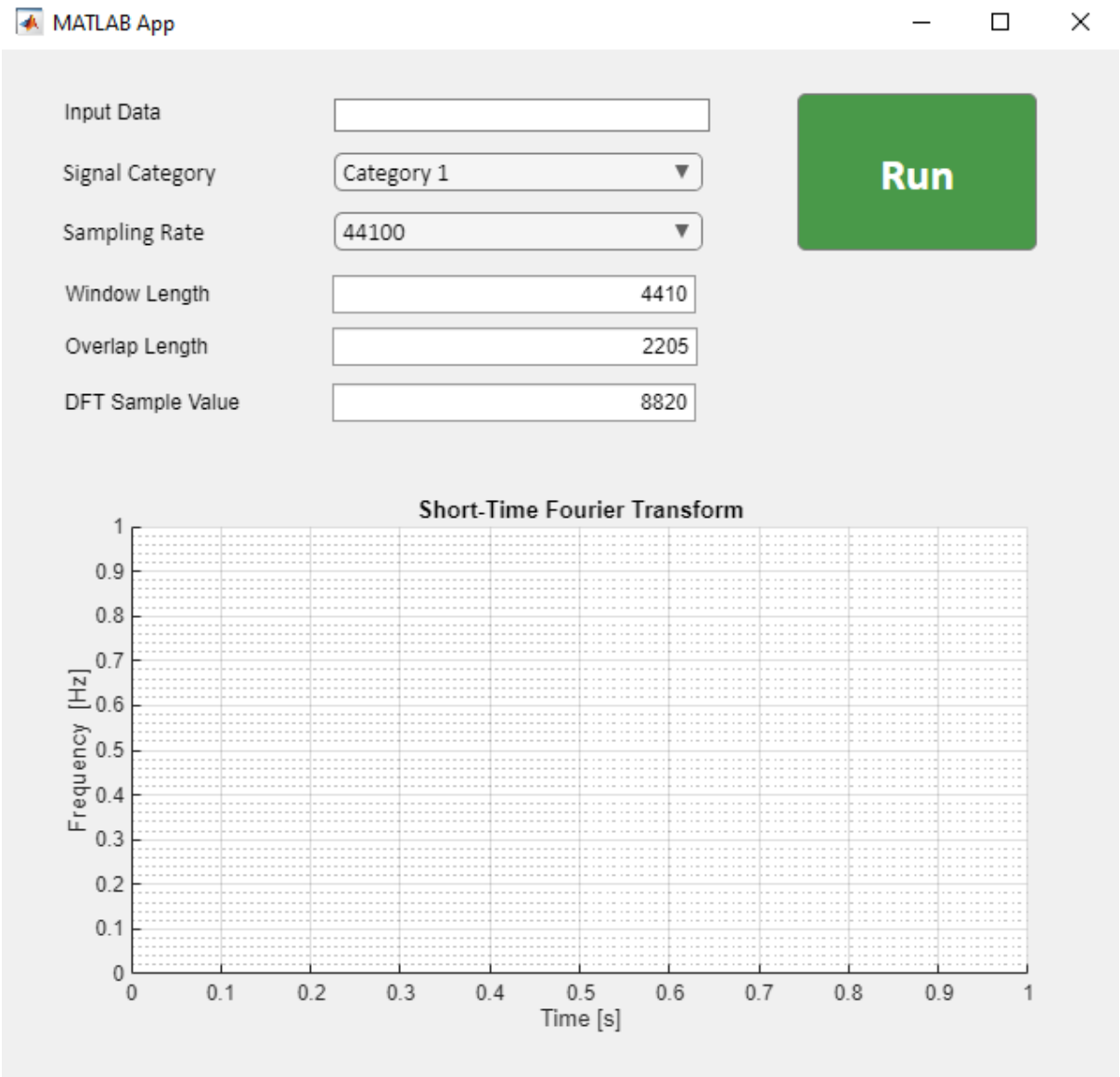

Figure 1: GUI for the transmitter program

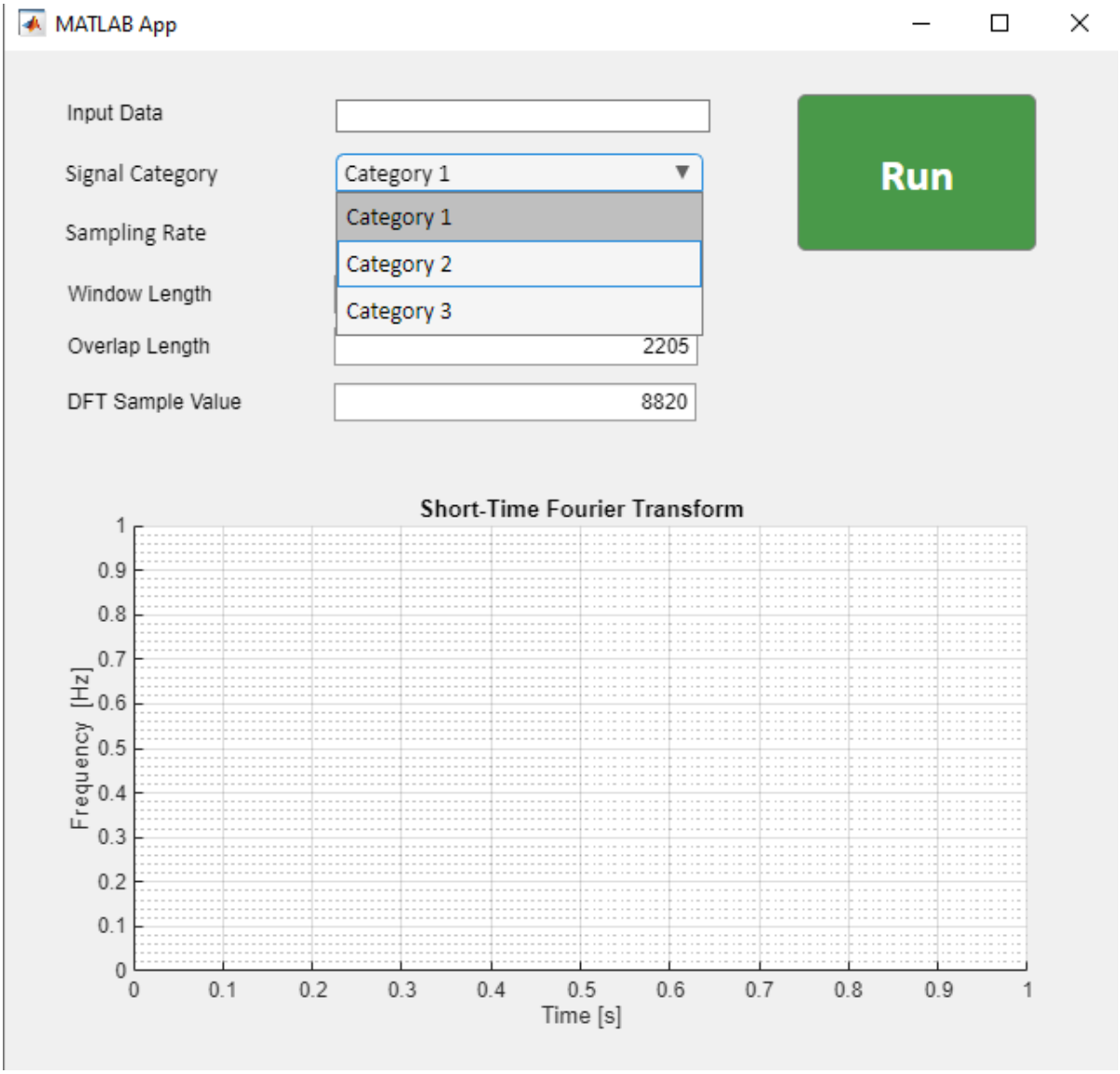

Figure 2: GUI for the transmitter program that shows Signals Category Selection

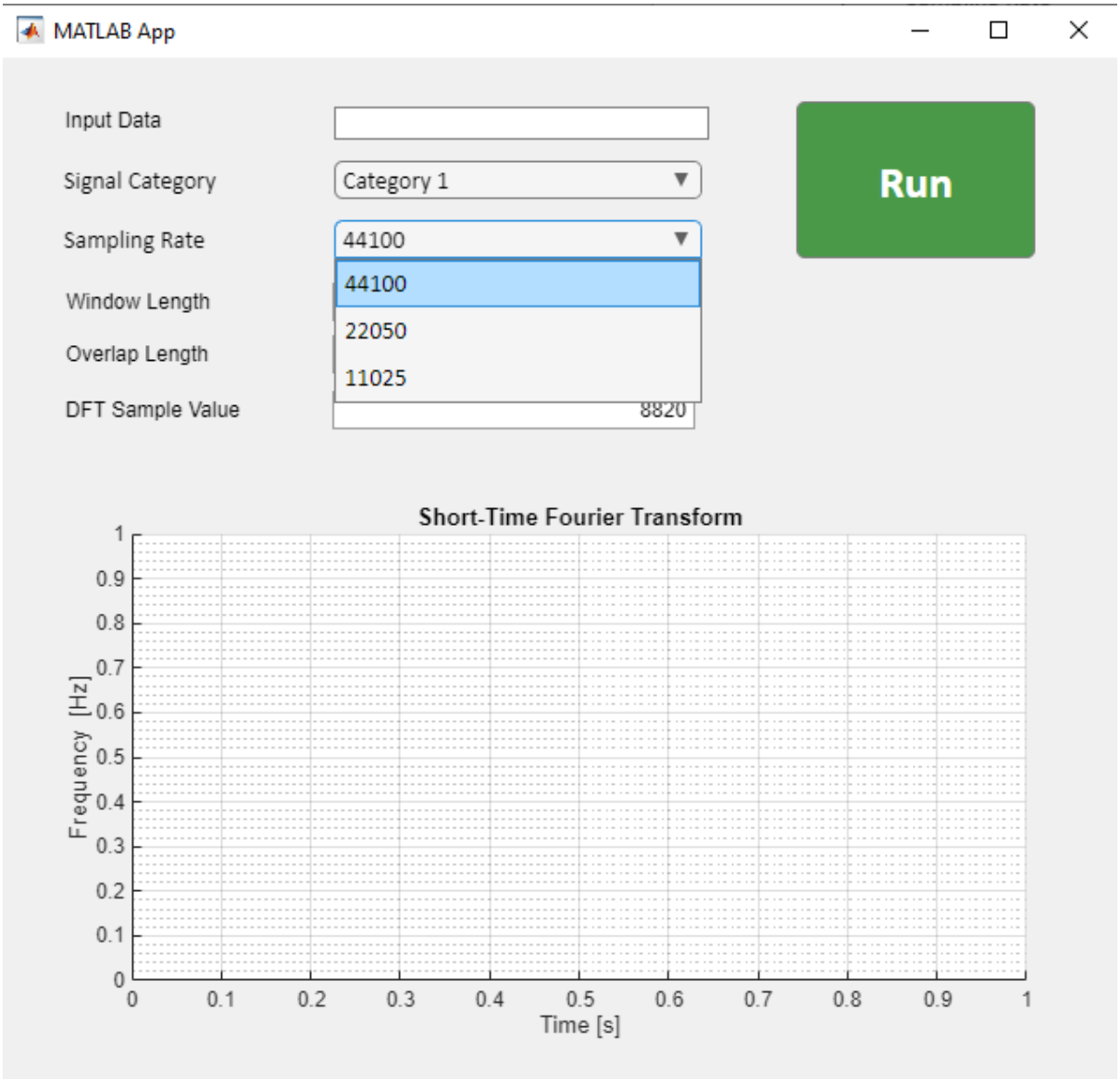

Figure 3: GUI for the transmitter program that shows Sampling Rate Selection

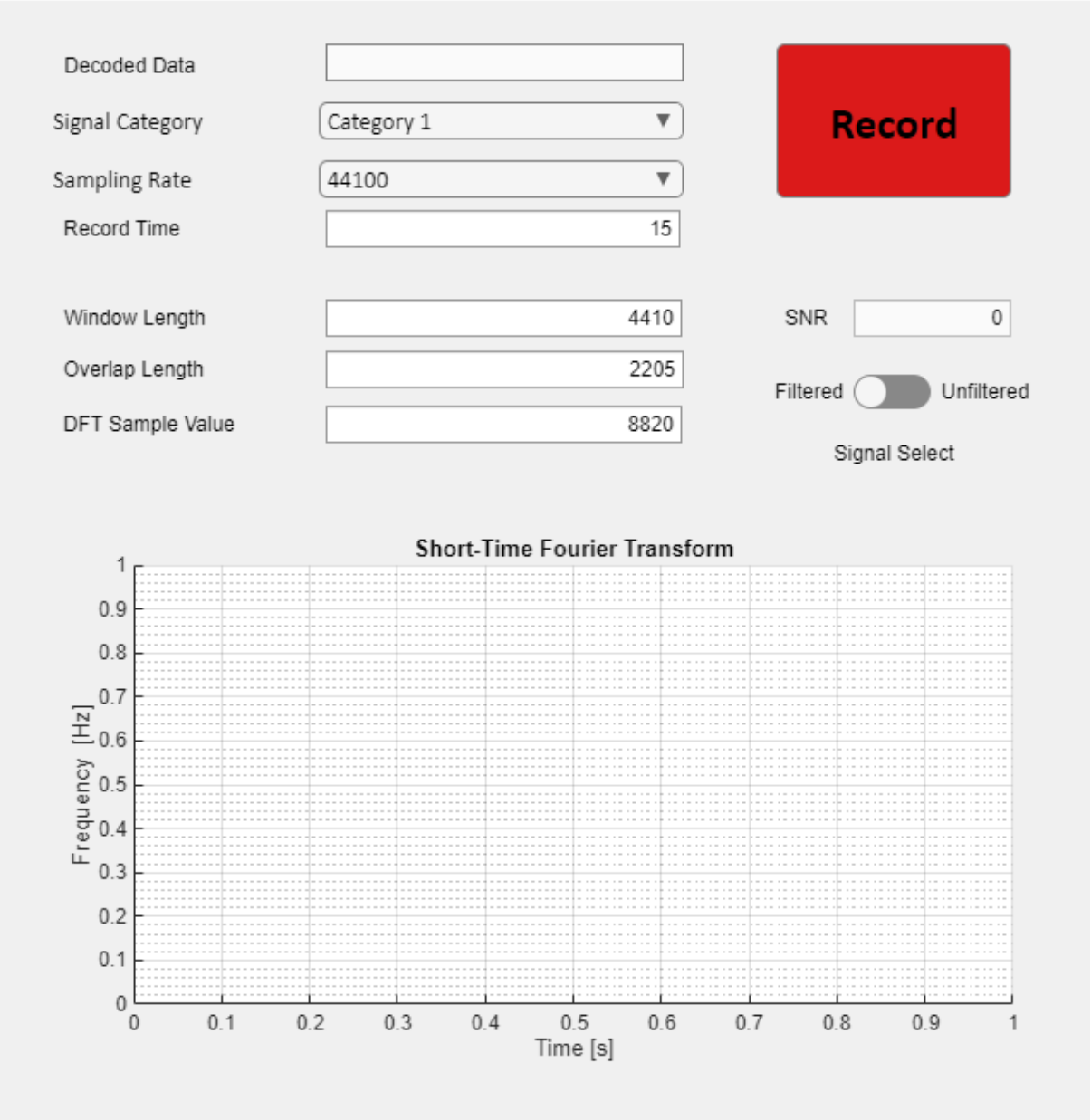

Figure 4: GUI for the receiver program

Spectrogram

The spectrogram is plotted using Short-Time Fourier Transform (STFT). It decomposes a signal into its time-frequency representation. The process is given.

1. **Signal Segmentation:** The signal is divided into overlapping frames (segments) of length M, with an overlap of L samples between consecutive frames. This ensures temporal continuity and minimizes boundary effects.
2. **Windowing:** Each frame is multiplied by a window function g[n] (in this case, a Bartlett window). This reduces spectral leakage by tapering the edges of the segment.
3. **Fourier Transform:** The discrete Fourier transform (DFT) of each windowed frame is computed, providing the frequency spectrum for that time segment.
4. **Spectrogram Visualization:** The squared magnitude of the resulting complex values is plotted on a time-frequency grid, representing power or intensity at each frequency over time.

Function used for calculating spectrogram of signal is given.

```
function [tint , fint , Xint] = plot_spectrogram(app,signal,fs)
      M = round(fs/10);   % defines window length M
      L = round(M*0.5) ;  % defines overlap length L
      g = bartlett(M);    % Window each segment with a Barlett Window
      Ndft = round(fs/5); % Evaluate the DFT point
      ts = 0:1/fs:(length(signal)/fs);
      segs = framesig(1:length(signal),M,OverlapLength=L);
      X = fft(signal(segs).*g,Ndft);
      framedT = ts(segs);
      tint = mean(framedT(2:end,:));
      fint = 0:fs/Ndft:fs-fs/Ndft;
      Xint = abs(X).^2;
      imagesc(app.Spectrogram,tint,fint,Xint);
      ylim(app.Spectrogram,[0,fs/2])
      xlim(app.Spectrogram,[0,length(signal)/fs])
end
```

M controls the time resolution (how finely we divide the signal in time) and frequency resolution (how finely we separate frequencies). Shorter M improves time resolution, capturing rapid changes in frequency. However, it sacrifices frequency resolution since fewer samples are available for Fourier Transform. Longer M improves frequency resolution but smears time resolution.

L defines how much the segments overlap. Higher overlap reduces artifacts and provides a smoother time-frequency plot but increases computational cost.

The Bartlett window reduces spectral leakage compared to a rectangular window by tapering the segment edges. This improves frequency localization at the cost of slightly broader peaks.

Ndft determines the frequency axis granularity. A higher Ndft adds more frequency points but requires more computational resources.

Mathematical explanation of STFT is given.

$$S(f,t) = \left| \sum_{n=0}^{M-1} x[n] \cdot g[n] \cdot e^{-j2\pi fn} \right|^2$$

Where $x[n]$ is the signal segment, $g[n]$ is the window function, f is the frequency. The choice of M, L, and Ndft directly affects the resolution:

- Time Resolution: $\Delta t = \frac{M}{Ndft}$
- Frequency Resolution: $\Delta f = \frac{Ndft}{M}$

Example Signals:

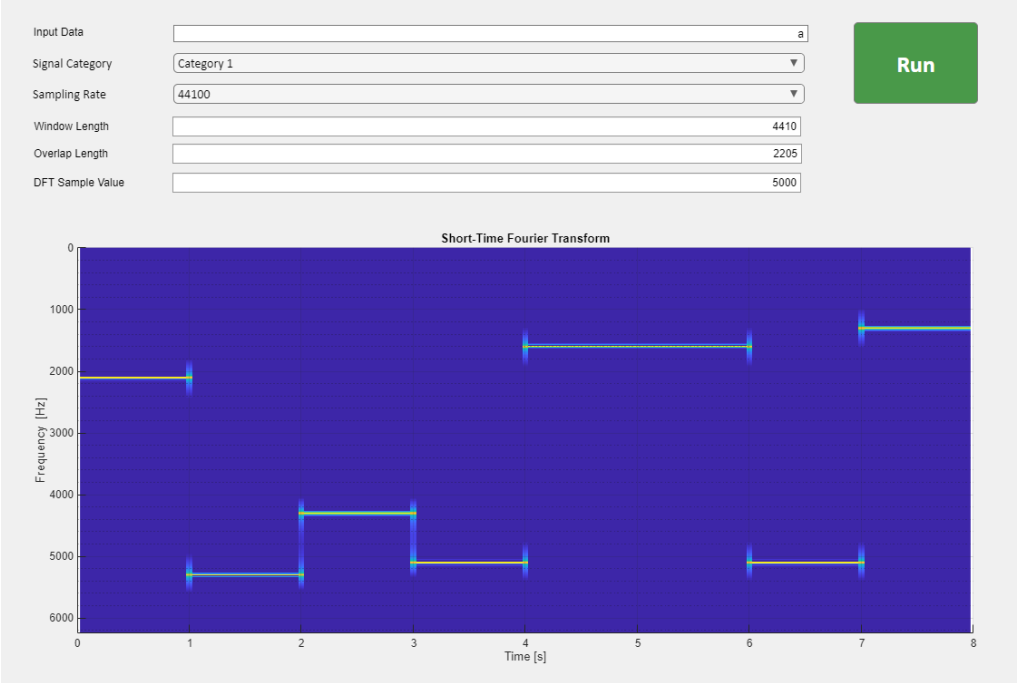

Figure 5

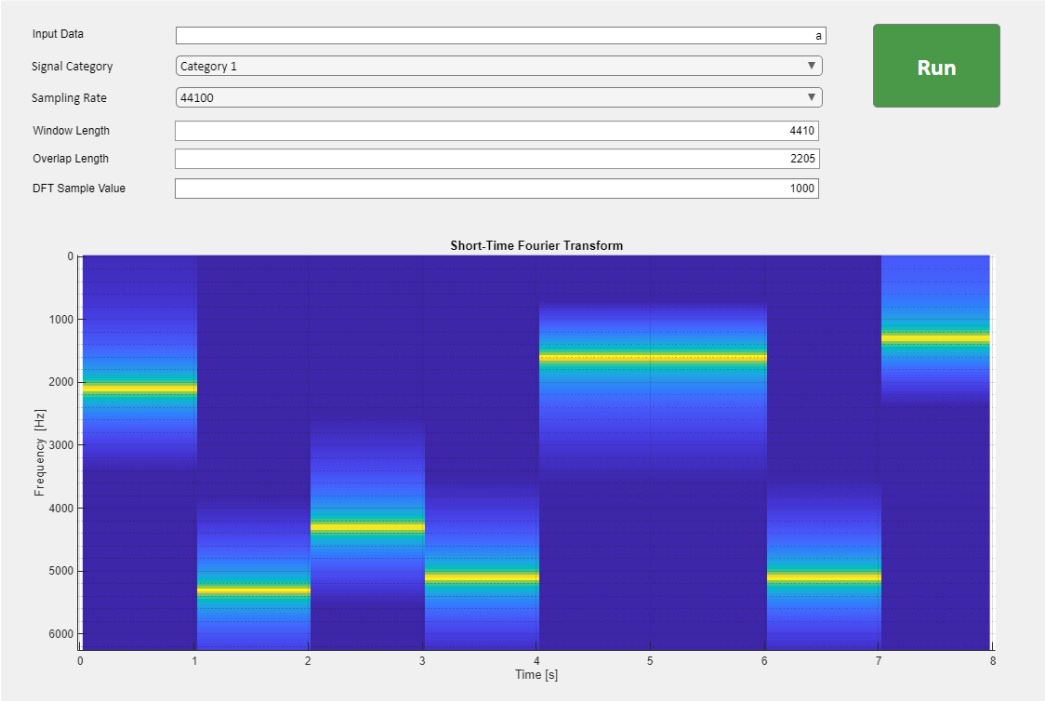

Figure 6

Encoding And Decoding The Text Message

Each character is first converted into ASCII values using MATLAB's "double()" function. This results in decimal values for each character. Decimal values are then converted into 8-bits by using the "dec2bin()"function and the resulting array is then reshaped into a row vector.

For category 1 signals, this binary data is read one at a time and using 2-FSK modulation, mk and then ∆f values are found. ∆f's are added to the hop-frequency for that time window which results in the modulated frequencies for the encoded data. Using these frequencies the message is generated and can be transmitted.

For category 2 signals the process is similar with the difference that binary data is read two at a time and 4-FSK modulation is used.

For category 3 signals the process is a little different since the binary data length is always multiples of 8 and this data can't always be read by 3 at a time. So a custom "padder()" function is implemented which takes the binary data stream and by adding 0's makes its length a multiple of 3. After this step the rest is similar to other category signals but now using 8-FSK.

This is why in the upcoming Figures 7. 8. and 9. show transmitted signals having 8, 4, 3 hopping-frequencies for category 1, 2, 3 respectively (for a single character transmission).

```
function binaryData = binaryConverter(app, inputData)
            binaryData = reshape(dec2bin(double(inputData), 8).',
1, []);
end
```

```
function padded_signal = padder(app, signalCategory, inputData)
            stringInputData = num2str(inputData);
            [~, inputdataLength] = size(stringInputData);
            if signalCategory == 'Category 1'
                divider = 1;
            elseif signalCategory == 'Category 2'
                divider = 2;
            elseif signalCategory == 'Category 3'
                divider = 3;
            else
                disp("Invalid Category!!")
            end
            padLength = mod(divider - mod(inputdataLength,
divider),   divider);

            zero_padding = repmat('0', 1, padLength);
            zero_padding
            padded_signal = [stringInputData , zero_padding]
            disp("Padded Signal")
            size(padded_signal)
            padded_signal
end
```

For decoding category 1 signals there is no need to remove the padding since there isn't going to be padding. When the signal is received and its length is detected by the SOI_finder function, an RNG is used with the same seed to select the expected frequencies from the frequency table. A demodulator function then uses these selected (expected) frequencies and compares them with the detected frequencies. Using these differences original binary data is obtained and then read as a text by the implemented "binarytoText" function. The function then returns the original text so the original signal is obtained.

```
function demodulated_data = demodulate(app, selectedFreqs, my_freqs)
           differences = my_freqs - selectedFreqs;
           [~, differences_length] = size(differences);
           demodulated_data = max(differences./100, 0);
end
```

```
function text = binarytoText(app, demodulated_data)
       bitGroups = reshape(demodulated_data, 8, []).';
       decimalValues = sum(bitGroups .* (2 .^ (7:-1:0)), 2);
       charArray = char(decimalValues);
       text = string(char(decimalValues)).join('');
end
```

Signal Generation

Category 1 Signal

Spectrogram of Category 1 Signal is given in figure. Assuming text message “a” is transmitted. Our program converts it to binary which is 01100001. For Category 1 message we expect to generate signals from tables [2200, 5200, 4200, 5200, 1700, 1700, 5200, 1200]. According to our data, we expect to see [2100, 5300, 4300, 5100, 1600, 1600, 5100, 1300].It can be seen in the figure we generated frequencies according to time of category 1.

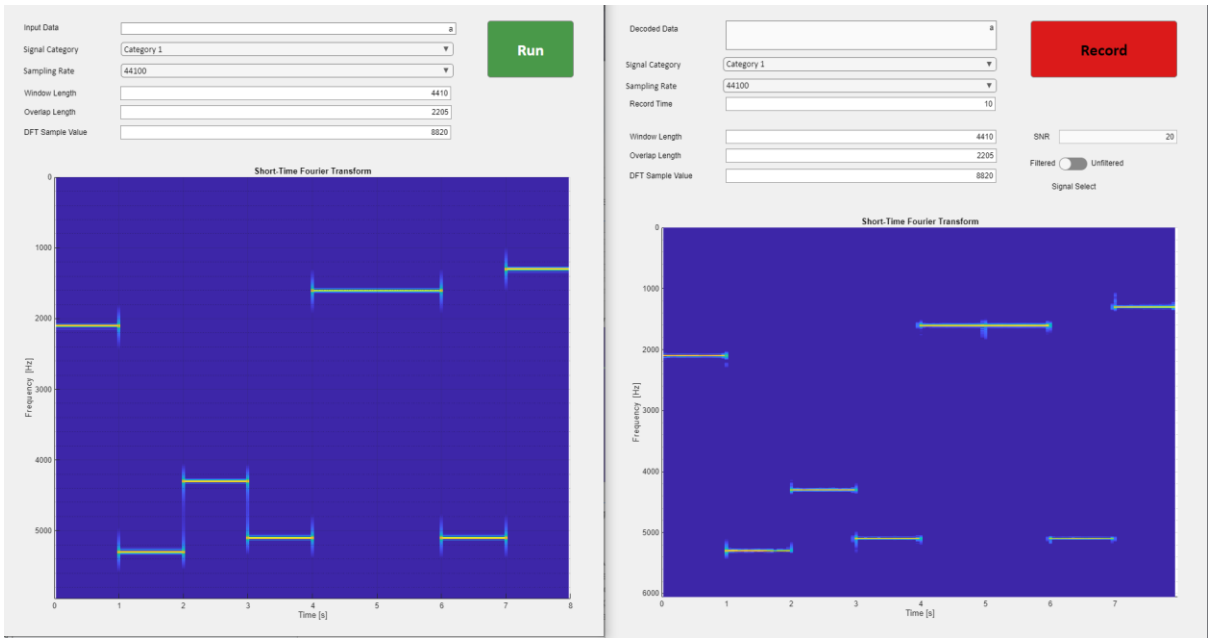

Figure 7: Category 1 signal spectrogram which input is “a”

Category 2 Signal

Spectrogram of Category 2 Signal is given in figure. Assuming text message “ab” is transmitted. Our program converts it to binary which is 01 10 00 01 01 10 00 10. For Category 2 message we expect to generate signals from tables [2800, 7800, 6800, 7800, 1800, 1800, 7800, 800]. According to our data, we expect to see [2650, 7950, 6500, 7650, 1650, 1950, 7500, 950].It can be seen in the figure we generated frequencies according to time of category 2.

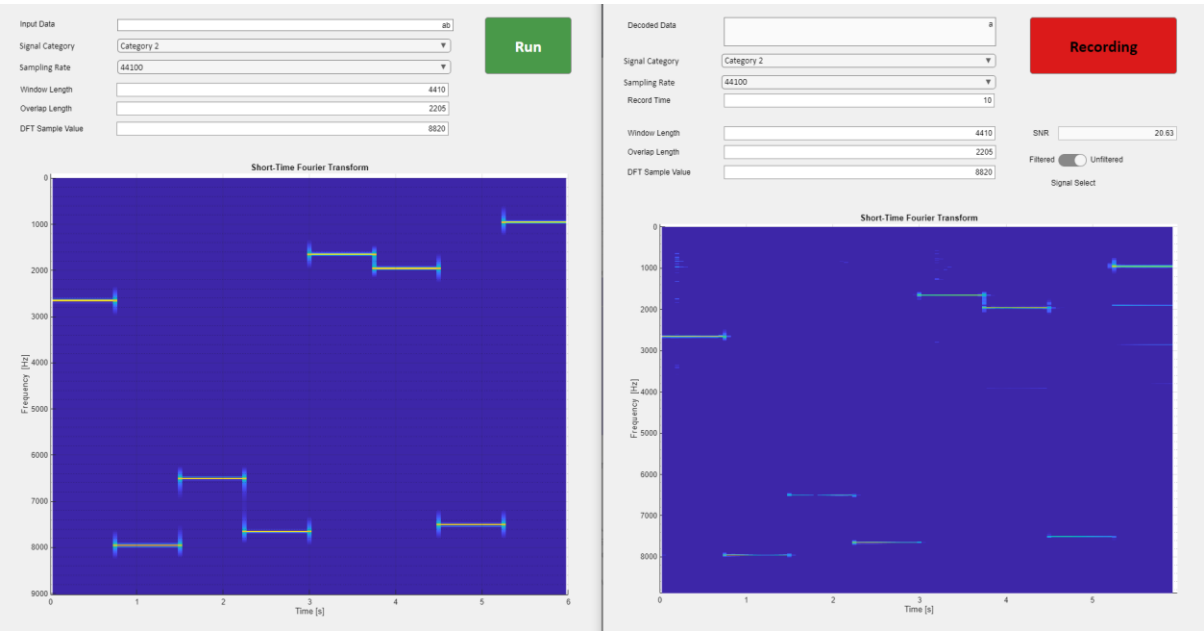

Figure 8: Category 2 signal spectrogram which input is “ab”

Category 3 Signal

Spectrogram of Category 3 Signal is given in figure. Assuming text message “abc” is transmitted. Our program converts it to binary which is 110 000 111 000 101 100 011. For Category 1 message we expect to generate signals from tables [3700, 11700, 9700, 11700, 3700, 1700, 11700]. According to our data, we expect to see [3500, 10900, 9300, 12300, 3100, 1100, 11900, 1500].It can be seen in the figure we generated frequencies according to time of category 3.

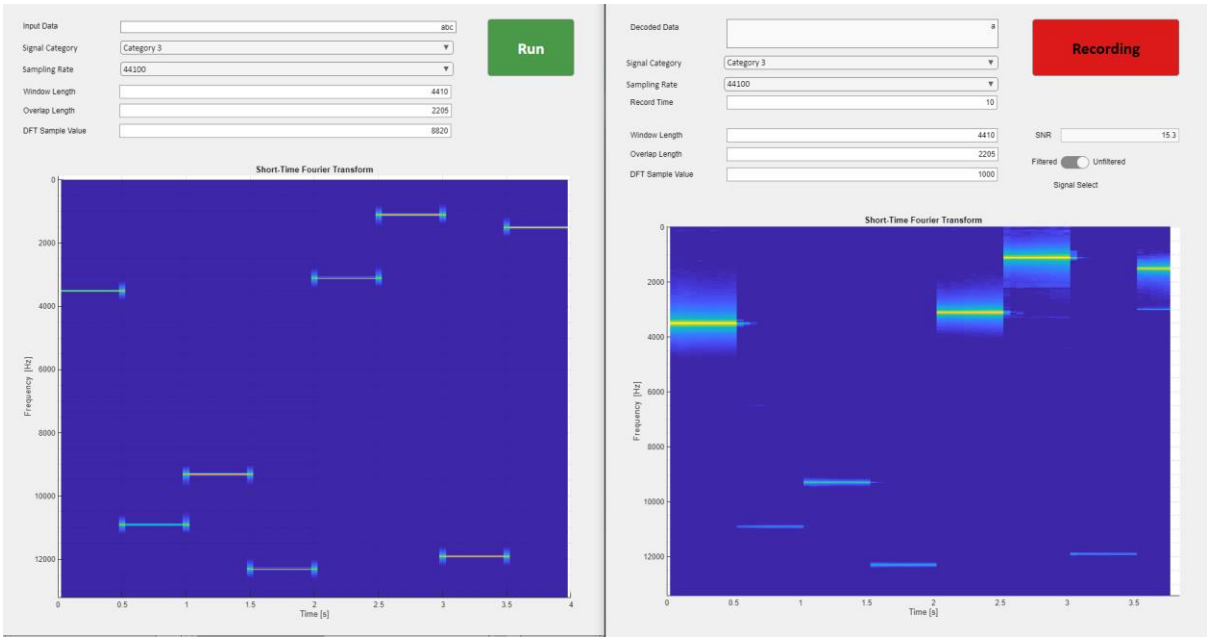

Figure 9: Category 3 signal spectrogram which input is “abc”

Received Signal Spectrogram

On the receiver side, the transmitted signal from the transmitter ,as illustrated in Figure 4, can be seen in Figure 7. The MATLAB ”audiorecorder” function to obtain signal. A Start-of-Interest (SOI) finder algorithm was implemented to determine the start and end points of the signal.

The SOI detection algorithm operates as follows:

1. **Bandpass Filtering**: firstly bandpass our signal between possible minimum frequency and maximum possible frequency.
2. **Signal Magnitude Computation**: We take the absolute value of the signal to process power data versus time.
3. **Low-Pass Filtering**: We apply a low-pass filter to remove small distortions in the absolute of the signal.
4. **High-Pass Filtering**: We apply a high pass filter to obtain fast increases and fast decreases.
5. **Threshold-Based Detection**: We detect the first increase and last decrease which are bigger than the determined threshold.
6. **SOI Boundaries**: We choose the first increase as the start of SOI and last decrease as the end of SOI.

After we obtain SOI. We obtain the length of the signal using the sampling rate and size of the signal array. We use this signal length to generate random numbers using RNG functions which are the same as transmitter side generated random numbers. Therefore, we can obtain expected frequencies. We use those frequencies and filter 150Hz much and 150Hz less. In that way we reduce the noise in frequencies our receiver does not expect.

After determining the SOI, the length of the signal is determined using the sampling rate and the size of the signal array. This method is implemented at the “findSignalLength” function. This signal length is used to generate random numbers using RNG. In that way we are ensuring consistency with the random numbers generated on the transmitter side because both rng functions have the same seed value. This method is implemented at the “selectFreqsfromTable2” function.

Using the generated random numbers, the expected frequencies are derived. These frequencies are then used to design bandpass filters with a tolerance of ±150 Hz around each expected frequency. This filtering process effectively attenuates noise in frequency bands outside the range of expected frequencies, thereby enhancing signal quality and reducing interference.

In figure 7 we see that unfiltered signal. In an unfiltered signal spectrogram we can see that there is a lot of noise between 1 to 4 seconds of SOI. But in the filtered signal which is demonstrated in figure 8, we can see that those noises are filtered.

Calculations to obtain spectrograms are the same as before.

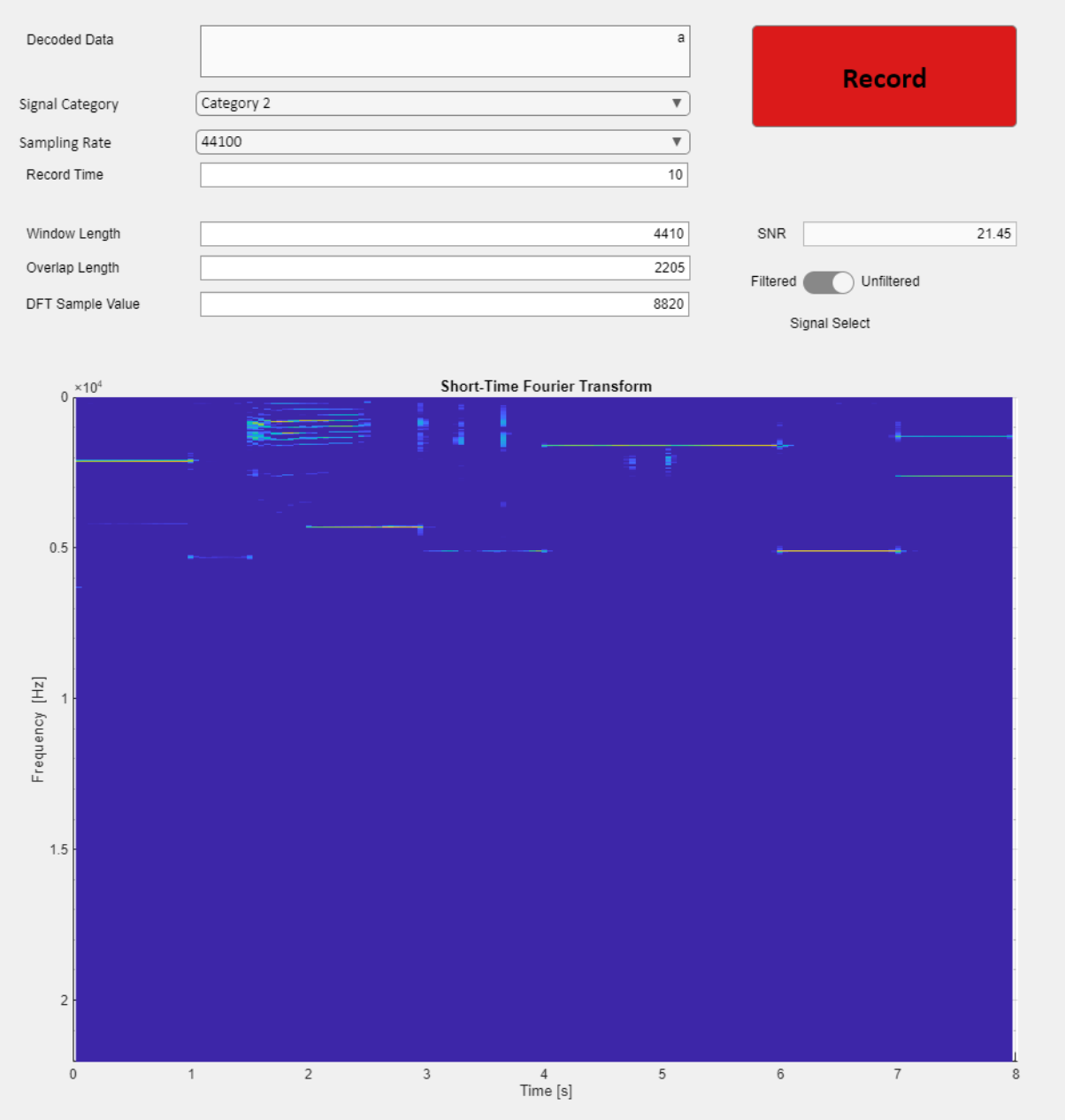

Figure 10: Received signal unfiltered

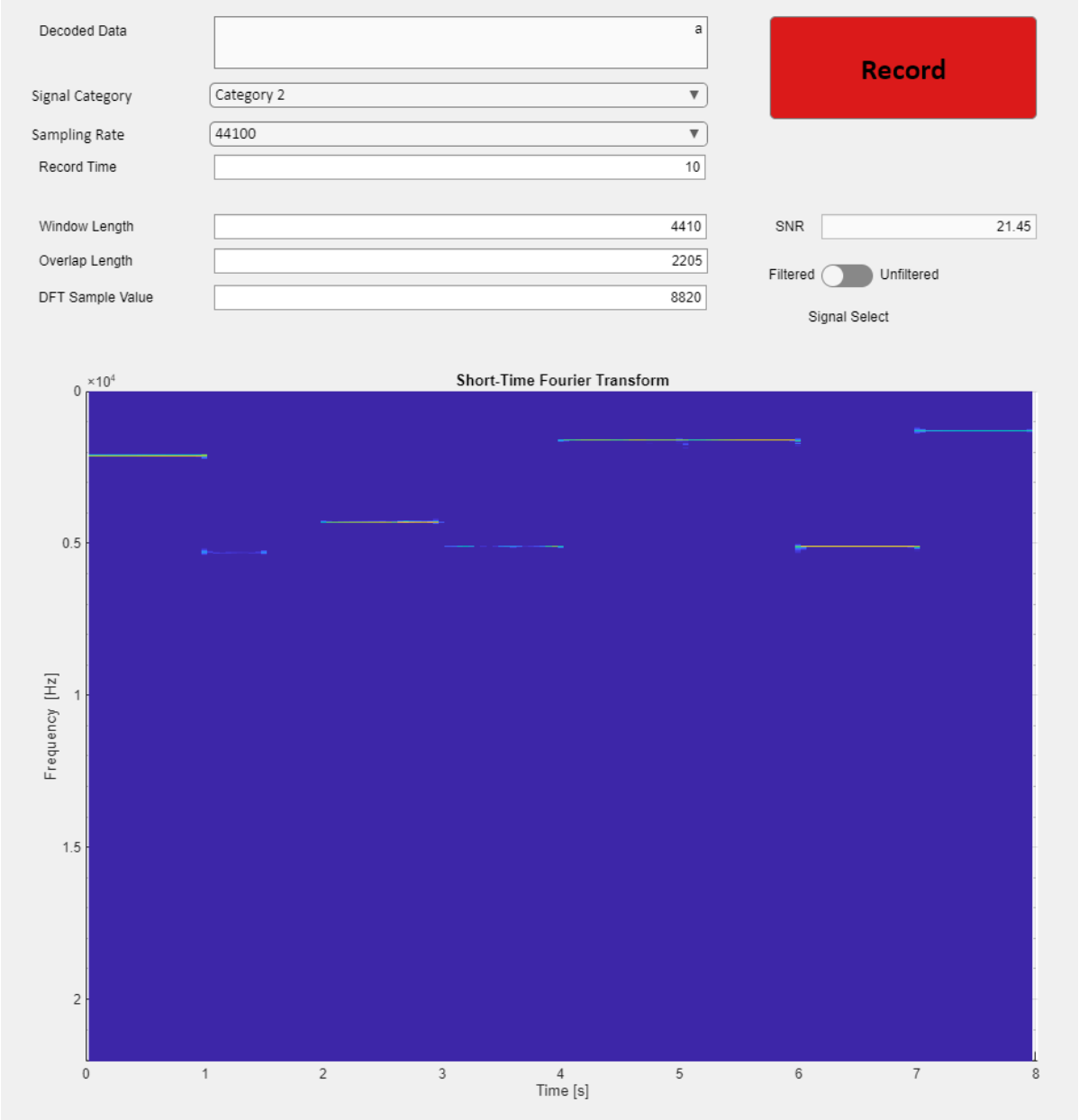

Figure 11: Received signal filtered

Tracking the Hop Frequencies

For category 1 signals it is known that there is one hop-frequency in each 1 second window. Knowing these and using the sampling rate, length of a signal can be found which corresponds to how many hop frequencies does the signal contain. Finding what these frequencies are is done by the “find_freqs()” function which is explained in the next section.

Demodulating Received Signal

After the filtered SOI is obtained its frequency spectrum is obtained. In this spectrum, for each hop period, the frequency that has the maximum amplitude is recorded. Since the recorded data is filtered with a bandpass filter centered around the expected frequencies, now the frequencies that have the maximum amplitude are the modulated frequencies from the transmitter. This functionality is implemented in the “find_freqs()” function.

(For Part I. find_freqs() is only implemented for Category 1 signals. But it can also easily be used for other categories if the parameter `interval_length` is changed for another category e.g. 0.5 for Category 2)

Now that each modulated frequency is obtained for each hop period, the difference between these obtained frequencies and the expected hop frequencies (obtained from the RNG of receiver side) gives the product: $\Delta f * mk$.

After this is done, the signal can be decoded by using “demodulate()” and “binarytoText()” functions.

Demonstration of Decoding a Category 1 Signal

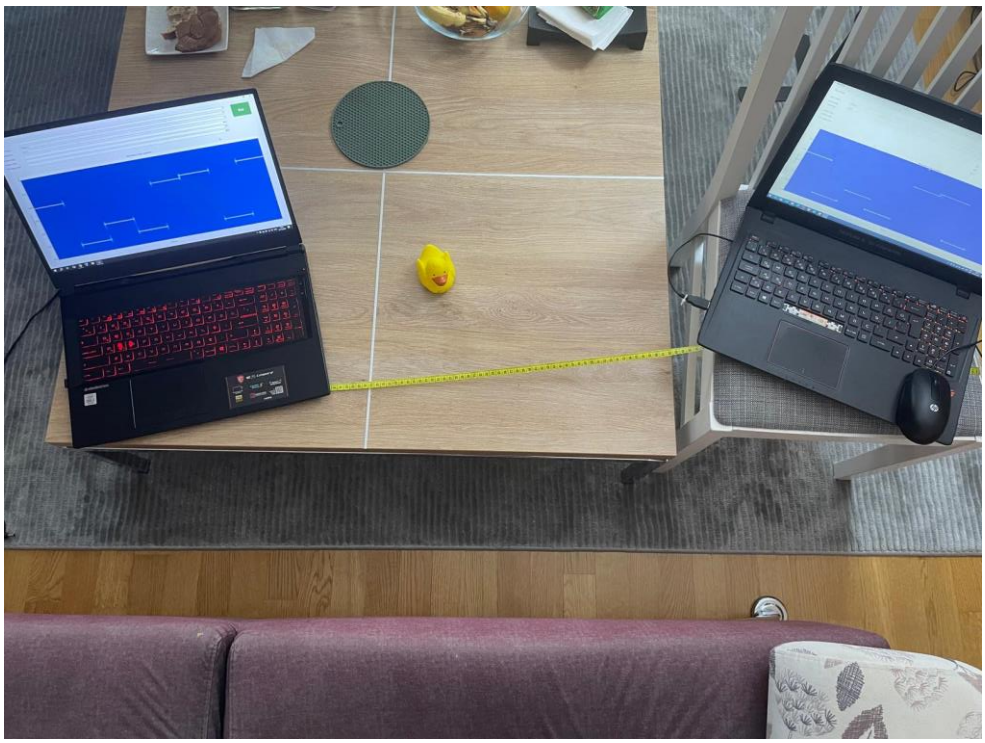

Figure 12

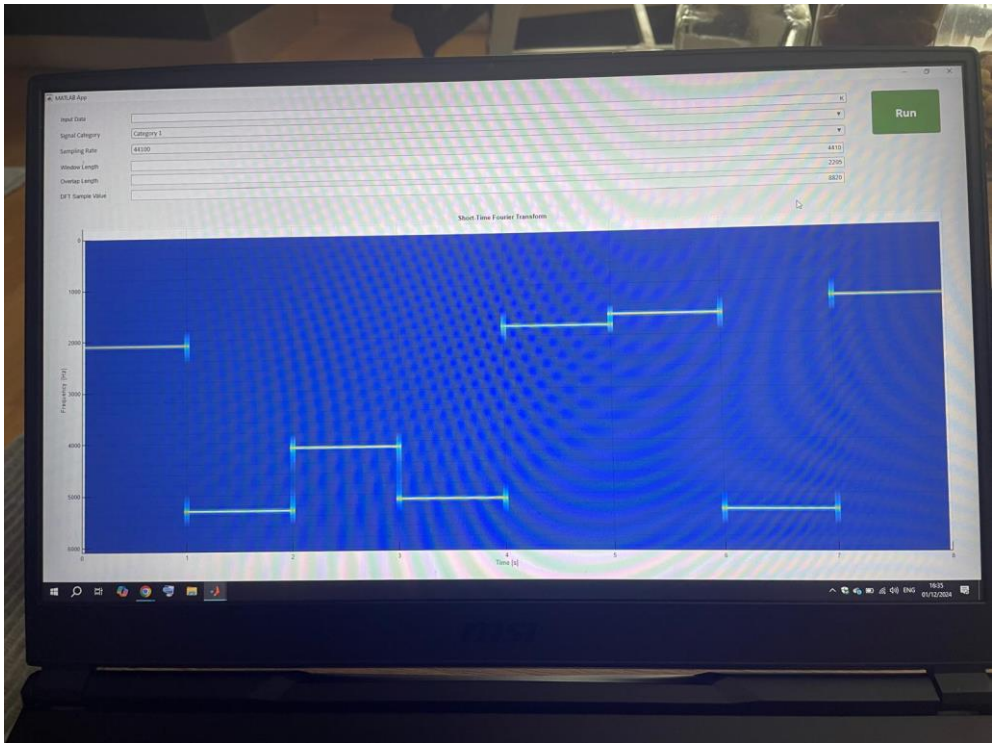

Figure 13

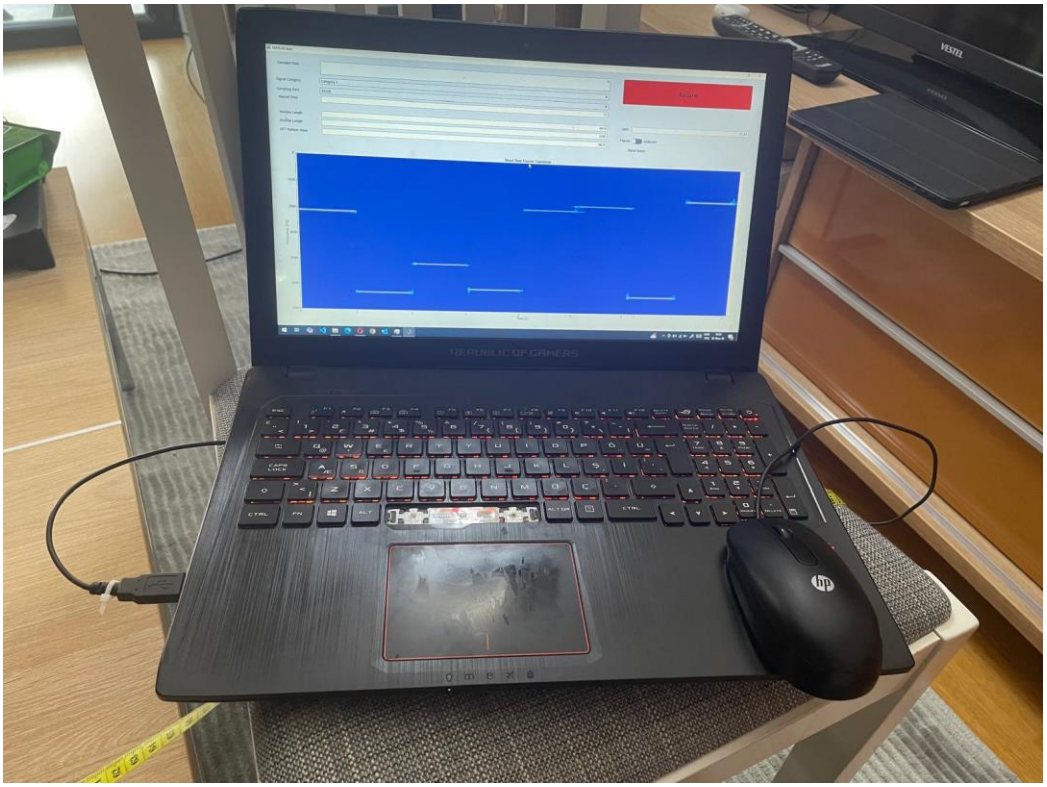

Figure 14

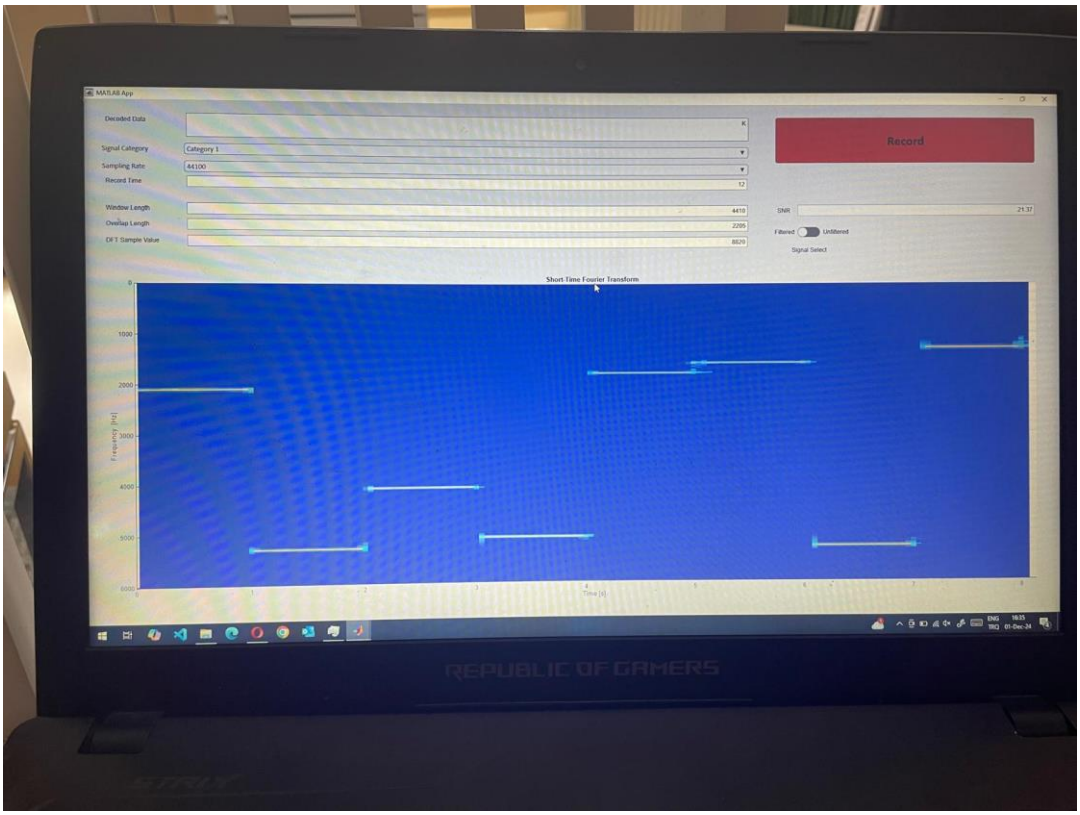

Figure 15