Berke Bayrak – 2442556
Emirhan Yılmaz Güney – 2443208
Yunus Emre Tüysüz – 2444032

# EE433 – Term Project Report

## Introduction

In this project, we will design and implement an FIR Wiener Filter specifically for noise suppression. Wiener filters are recognized for being theoretically optimal in minimizing mean square error (MSE). However, the application of a Wiener filter requires a priori knowledge of the input signal, or the noise involved. To address this, our project includes estimating the environmental noise type, focusing on two primary candidates: Pink noise and White noise. After identifying the type of noise and its standard deviation, this critical information will be used to correctly apply the Wiener filter. The entire signal processing operation will be conducted within a C function, integrated into a LabVIEW block diagram that collects speech inputs via a microphone and outputs the processed signal through a speaker.

## Theoretical Background

As already mentioned in the Introduction part, Wiener filters are MSE optimum filters and we need autocorrelation of the input signal and cross correlation of input and desired signals to be able to apply Wiener-Hopf equation given as

$$R_x h = r_{dx}$$

where $R_x$ is the autocorrelation matrix of the input and $r_{dx}$ is the cross-correlation vector between the desired signal and the input x.

In this project, the target or desired signal will be isolated speech. We assume that the noise and speech signals are uncorrelated. Under this assumption, we will have

$$r_x = r_v + r_s$$

Hence, we can find $r_s$ by subtracting $r_v$ from $r_x$. Also, $r_{dx}$ will be equal to $r_s$ due to uncorrelated x and s. As for $R_x$ , it will be constructed by arranging the elements of $r_x$ into a Toeplitz matrix form.

To be able to calculate $r_x$ , we will be estimating it with a sample autocorrelation as follows

$$\hat{r}_x[k] \approx \frac{1}{N} x[n] * x^*[-n] = \frac{1}{N} \sum_{m=0}^{N-1} x[m]\, x^*[m-k]$$

Hence, FIR Wiener filter coefficients will be found as

$$h_{opt} = R_x^{-1} r_{dx}$$

In terms of noise estimation, distinguishing between White and Pink noise is crucial. Theoretically, White noise exhibits a unique autocorrelation characteristic, where correlation exists only at zero lag, located as an impulse at zero. This contrasts with Pink noise, which maintains correlation across its samples. The defining characteristic of Pink noise in the frequency domain is its power spectral density (PSD), which is inversely proportional to frequency ($^1/_f$). The PSD represents the Fourier Transform (FT) of the autocorrelation function. Given the similar structure of their autocorrelation functions considering a large lag spectrum, distinguishing these two types of noise becomes more practical and accurate in the frequency domain. This differentiation is essential for the accurate application of noise suppression techniques in our project.

## Implementation

For the implementation of noise detection, we have used a block processing approach. By calculating the energy of blocks and comparing them to an adaptive threshold, which is determined by the average energy of the last five blocks, we have detected blocks which only contain noise segments. Using these noise segments, standard deviation of the noise can be estimated easily. As for the estimation of noise type, we have calculated the PSD of these detected noise segments. After calculation of PSD in log-scale, a linear regression model is applied to fit a line to PSD of these noise segments. If PSD has a flat fit, it is estimated as White noise. On the other hand, it has a slope larger than -7.5dB/decade, it is estimated as pink noise. This threshold is set by considering the $^1/_f$ characteristic of the PSD of the Pink noise. It is important to note that some speech blocks may be misidentified as noise blocks, especially in scenarios with very high signal-to-noise ratios (SNR). While this method is not flawless, it provides a functional and effective means for estimating noise characteristics.

After estimation of noise type and standard deviation, Wiener filter is implemented by using this information since autocorrelation of White and Pink noise are already known. When calculating the Wiener filter coefficients, regularization is performed to improve the performance of noise filtering. Regularization constant, $\boldsymbol{\lambda}$, is arranged by fine tuning using the estimated standard deviation of the noise. Depending on the noise type, only autocorrelation function is updated, and the rest is the same.

This project is implemented in three different environments. The first environment is Matlab, which served as the primary platform for developing, tracking, and visualizing the implementation process. Matlab's extensive built-in functionalities and its visualization tools made it ideal for prototyping and refining our system.

The second one is the C programming language. The code written in Matlab is translated into a C code since the project will eventually run in LabVIEW environment. During the translation process into C, some of the functions could not be implemented as accurately as Matlab since C does not have that much built-in functionality opportunities. Hence, there is

some performance loss, in terms of accuracy of results, when translating into C. For instance, FFT and matrix inversion operations need to be implemented manually. FFT is implemented using a recursive divide-and-conquer approach that splits the input into even and odd indexed elements, computes FFT on these smaller parts, and then combines them using the discrete Fourier transform (DFT) formulas for even and odd terms. This manual implementation, while functional, might not achieve the same level of optimized performance or precision as specialized libraries like FFTW in C or Matlab's built-in functions, which utilize more complex algorithms and hardware acceleration features. As for the matrix inversion operation, LU decomposition is used to factorize the given matrix A into lower L and upper U triangular matrices. This factorization facilitates solving sets of linear equations for each column of the identity matrix using forward and backward substitution, which effectively computes the columns of the inverse matrix inv(A). This approach is manually implemented and requires careful management of memory and error conditions, such as division by zero, which may not be as robustly handled as in higher-level languages or libraries.

The final one is LabVIEW environment, where all the processing is performed eventually. LabVIEW utilizes the C function to be able to perform processing. Other than this, it operates as an interface to collect sound data and play it. Again, during the implementation of LabVIEW, we have lost some accuracy due to limitations of the environment. For instance, there should be a small delay between microphone input and audio output which causes clicks in the output. This problem is resolved by overlap and add method; however, it still introduces some errors and causes accuracy to decrease. Hence, while the project works best and “almost” perfect in Matlab environment, there are some distortions in LabVIEW.

## LabVIEW Block Diagrams and Front Panels

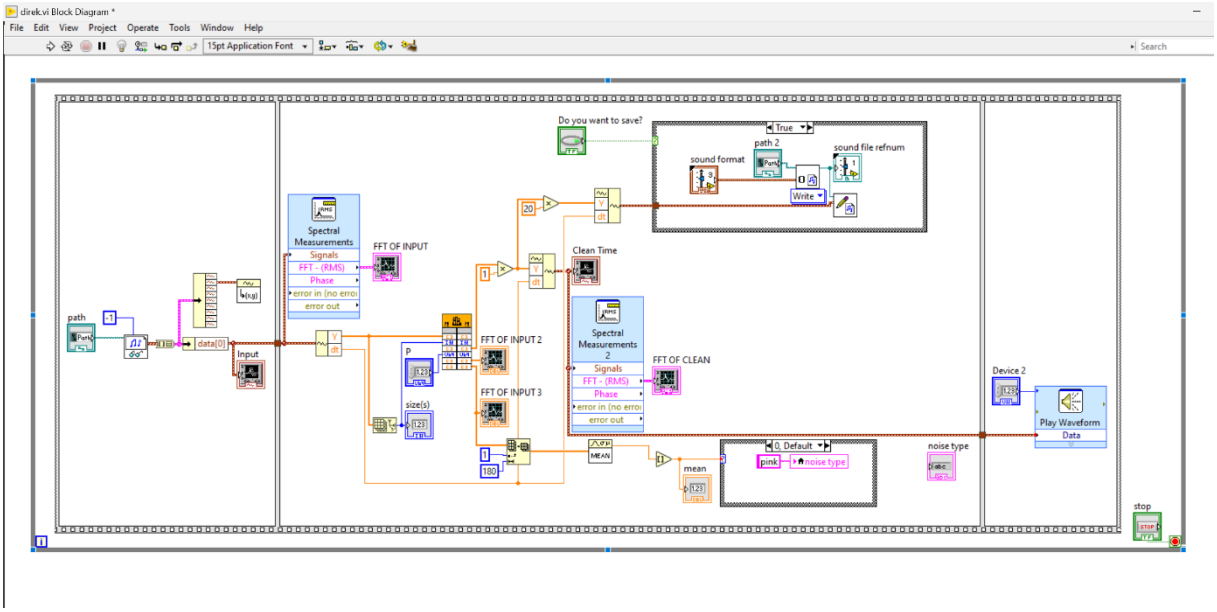


Figure 1. Block diagram of the VI used to import sound file

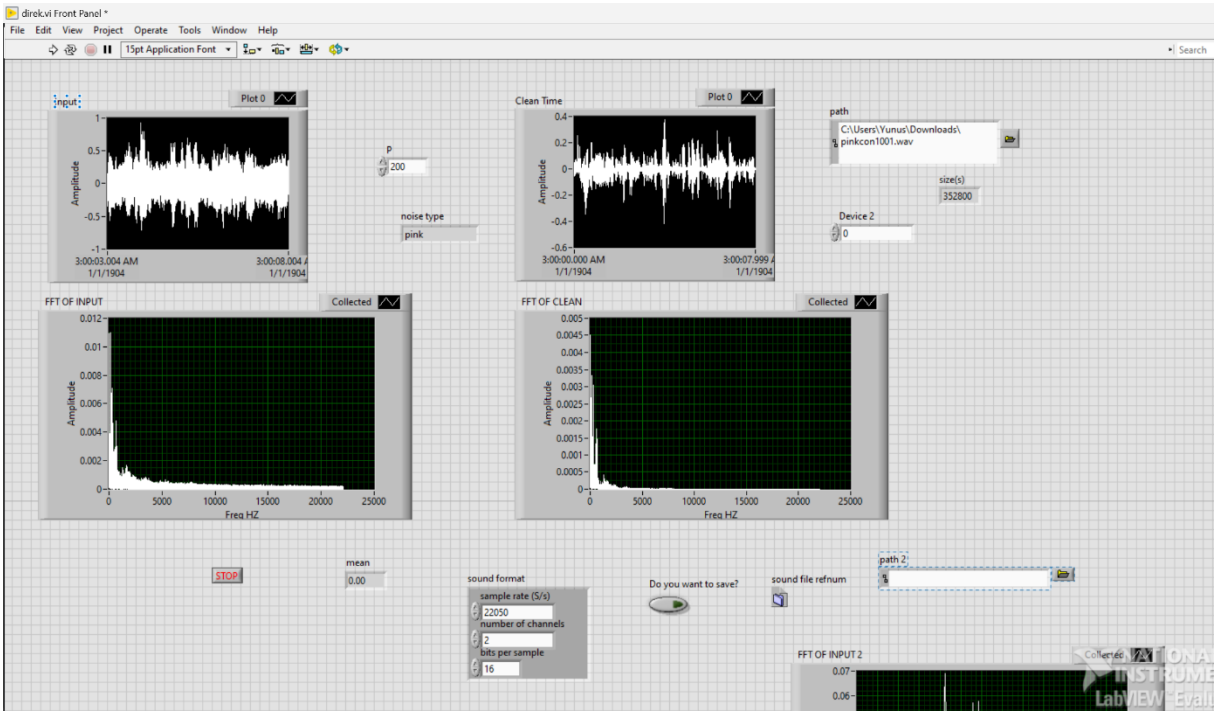


Figure 2. Front panel of the VI used to import sound file

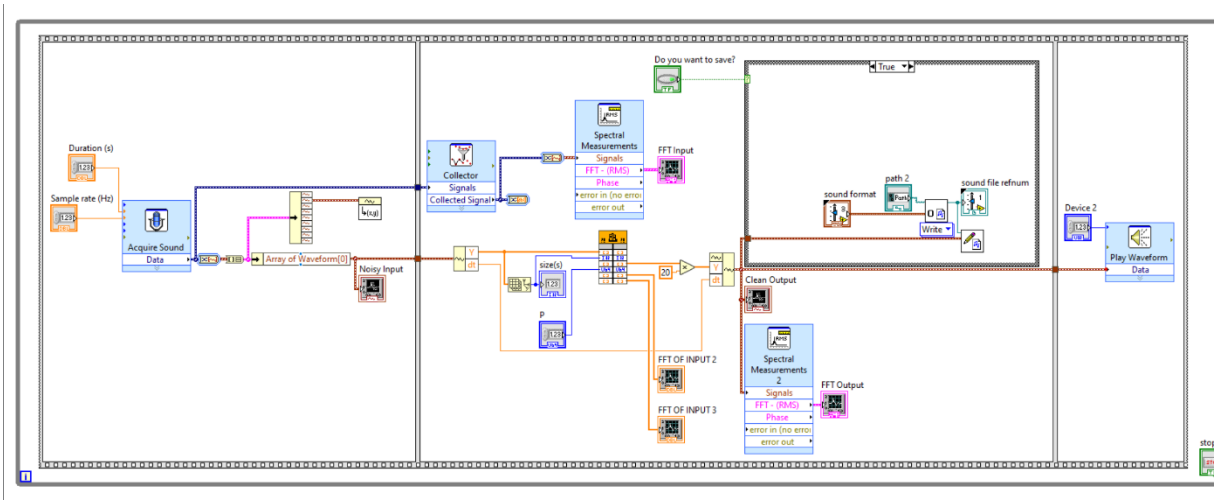


Figure 3a. Block diagram of the VI used to get sound from microphone

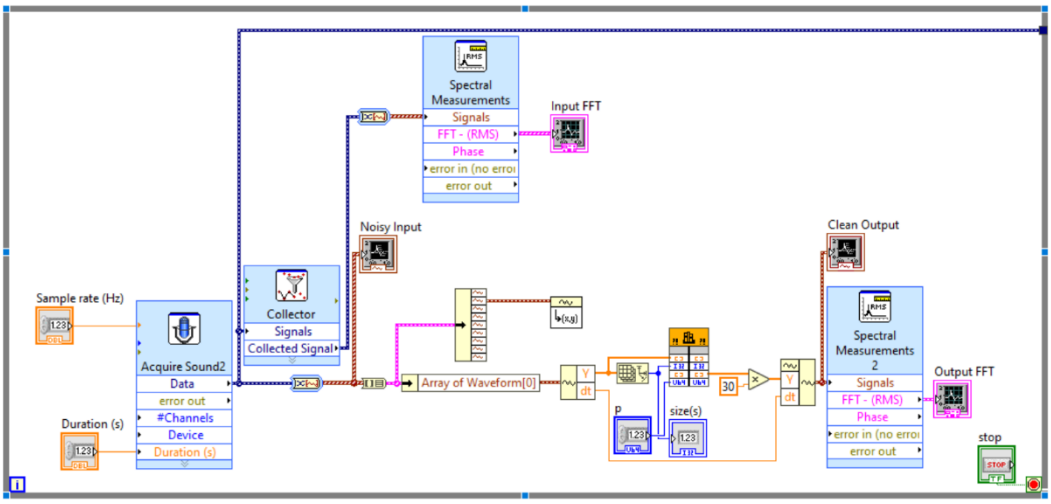


Figure 4a. Block diagram of the VI used to test real time

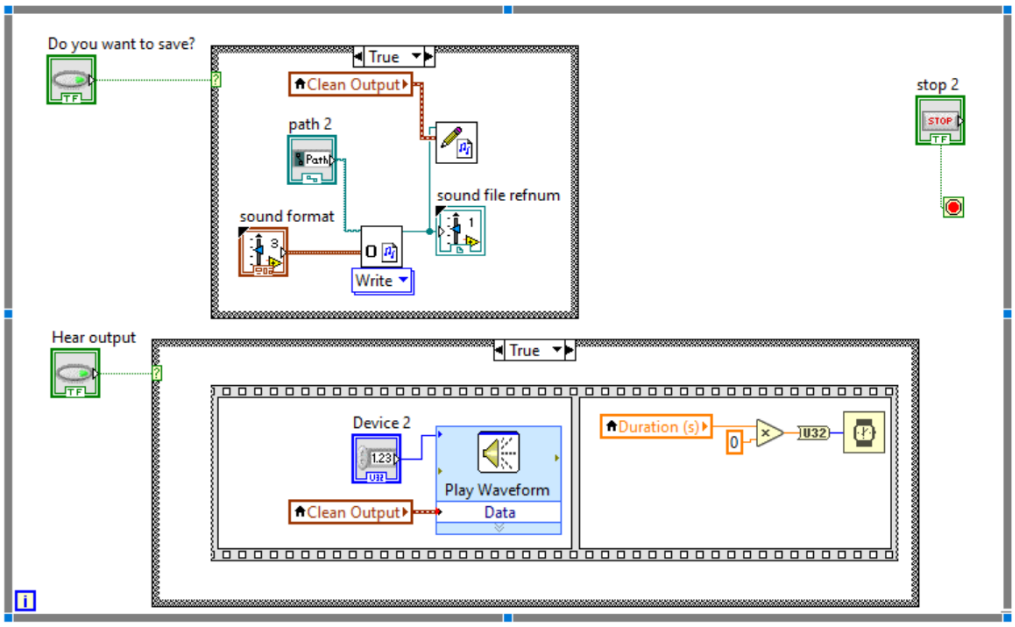


Figure 4b. Block diagram of the VI used to test real time

## Results and Discussion

As mentioned in the previous sections, while the implementation in Matlab is almost perfect, the implementation in LabVIEW has some defects and a more limited range for input SNR values. While Matlab can recover signals corrupted with noises with 0.7 standard deviations, LabVIEW can merely recover from 0.5 standard deviations.

### Test Procedure

In the Test Procedure, a prerecorded speech is corrupted with a desired noise. Either SNR or standard deviation can be given as input to the system to generate noise. Then, this prerecorded speech is filtered and output is observed. This procedure works quite well and LSE is in the order of $10^{-4}$.

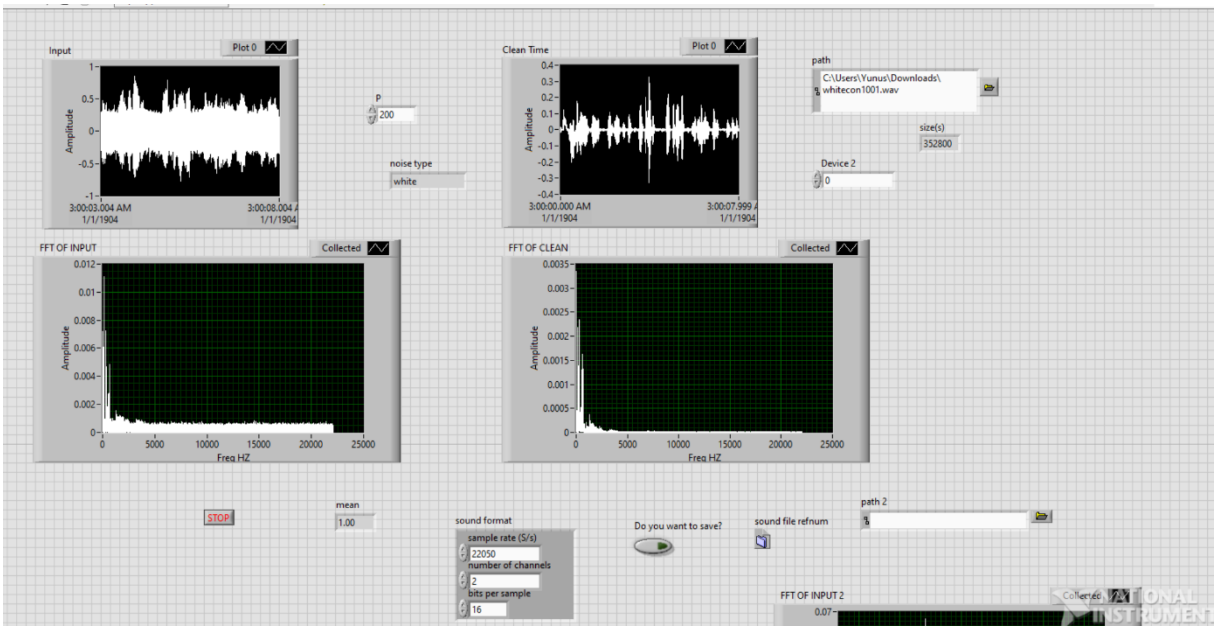


Figure 5. Filtering a pre-recorded noisy speech

As shown in Figure 5, speech is recovered from noise. This can be observed both in time domain and frequency domain. While we can clearly observe the effect of white noise in FFT of the input signal as a constant spectrum at high frequencies, it is cleaned in the FFT of the output signal. A filter length of 200 is used to perform this noise removal. This VI can be used to import sounds which are corrupted with noise in Matlab. Also, the resultant speech can be recorded by using “Do you want to save?” button. As observed in Figure 5, noise type is detected correctly.

### Run-time Mode

In run-time mode, we do not have any delay between input and output in Matlab, so the processing is performed in real time. On the other hand, we do have some delay, approximately 1 second, between input and output in LabVIEW, which is due to the fact that microphone and speaker cannot work simultaneously in LabVIEW. Hence, we make microphone wait for 60 ms every 1 second. This delay is adjusted by tuning what we have observed at the output. If we increase this delay too much, it is not real time anymore. On the other hand, if delay is decreased too much, we observe heavy clicking effects and the output is distorted extremely. This tradeoff is mainly due to LabVIEW environment. Thus, considering both the operation of C function, and the delay for microphone, there is a delay of approximately 1 second between input and output.

## Conclusion

In conclusion, we have completed this project both in Matlab and in LabVIEW, which utilizes a C function. In Matlab, both run-time and test procedure work completely fine. However, there are some distortions in LabVIEW as mentioned above many times. All in all, we have completed this project successfully and we believe that we are ready to be graduates of signal processing option.