NANYANG POLYTECHNIC
EG431D Data Acquisition

Lesson 5

Course: Diploma in Engineering (Electronic and Digital Engineering)

Module: EG431D Data Acquisition

Title: Pre-Processing and Filtering

Objective:

The objective of this lesson is to equip learners with an in-depth understanding of signal preprocessing and filtering techniques used in data acquisition systems. Emphasis will be placed on the purpose and processes involved in these techniques, including their role in reducing noise and preparing signals for subsequent analysis. Learners will gain practical knowledge on implementing preprocessing methods to smooth noisy signals while preserving critical information, as well as exploring the significance of various filtering approaches for isolating relevant signal frequencies tailored to specific real-time applications. Additionally, learners will understand how these techniques enhance the accuracy of signal detection and feature extraction in domains such as audio processing, communication systems, and biomedical signal analysis.

Learning Objectives:

- ❑ Describe the purpose and process of common signal pre-processing techniques in data acquisition systems and explain how the methods help in reducing noise and preparing signals for further analysis.
- ❑ Demonstrate the implementation of common signal pre-processing techniques and explain how these techniques smooth noisy signals while retaining important information for analysis.
- ❑ Describe the significance of different filtering techniques in isolating relevant signal frequencies for specific applications in real-time signal processing systems.
- ❑ Demonstrate how various pre-processing and filtering techniques can be applied to enhance the accuracy of signal detection and feature extraction in domains like audio processing, communications, and biomedical signal analysis.

1. An Overview of Digital Signal Pre-Processing and Filtering

Digital Signal Pre-Processing and Filtering are essential steps in data acquisition and analysis systems, providing the foundation for accurate and meaningful signal interpretation. In practical applications, raw signals obtained from sensors or other sources are often contaminated with noise, artifacts, or unwanted variations that can obscure valuable information. Signal pre-processing focuses on refining the raw data, enhancing signal quality, and preparing it for further analysis, while filtering specifically targets the isolation or removal of undesired frequency components to retain the most relevant signal features.

Pre-processing involves a range of techniques such as **Normalization**, **Scaling**, **Smoothing**, and **Transformation**. For example, normalization ensures that signals from different sources are brought to a common scale, which is crucial in multi-sensor systems where data comparability is required. Smoothing techniques, such as **Moving Average** or **Gaussian Smoothing**, reduce abrupt variations or noise, creating a more stable representation of the signal. These pre-processing steps are especially critical in applications where subtle variations in the signal, such as in biomedical signal analysis or seismic data interpretation, need to be preserved and highlighted for better decision-making.

Filtering, on the other hand, addresses specific frequency components of the signal. Figure 1A depicts an example of a signal heavily contaminated with noise is recovered by using a filter.

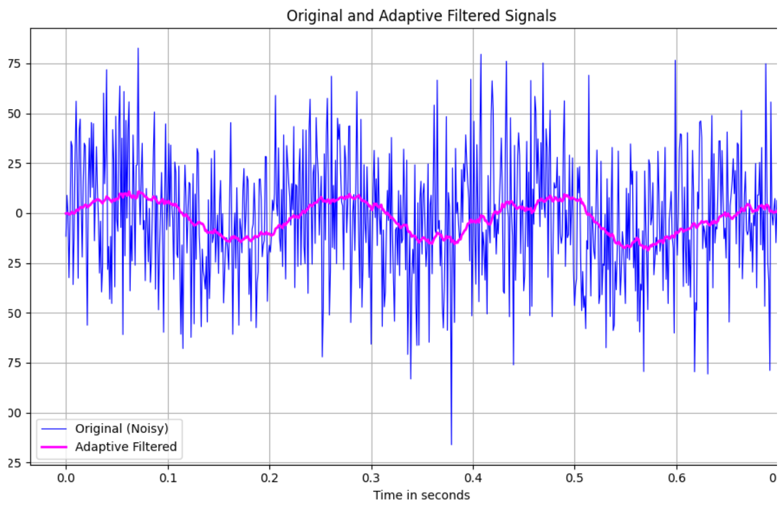


Figure 1A – A heavily contaminated signal is recovered by using a filter

Filters can be categorized as **Low-pass**, **High-pass**, **Band-pass**, or **Band-stop**, depending on the application. Low-pass filters allow signals below a certain frequency to pass while attenuating higher frequencies, making them suitable for removing high-frequency noise in slowly varying signals. Conversely, high-pass filters eliminate low-frequency noise.

Figure 1B (a) illustrates the application of a low-pass filter to isolate a low-frequency signal from high-frequency noise. The first subplot shows the original low-frequency signal, which represents the desired component of the data. The second subplot combines this low-frequency signal with high-frequency noise at 50 Hz, simulating a real-world noisy environment. The third subplot displays the filtered result after applying a low-pass filter with a cutoff frequency of 20 Hz. The filter effectively removes the high-frequency noise while preserving the original low-frequency signal, demonstrating its utility in enhancing signal quality for further analysis or processing.

The plot in Figure 1B (b) demonstrates the frequency response of a 4-th order Butterworth low-pass filter, designed to allow low-frequency signals to pass while attenuating higher frequencies. The blue line represents the magnitude response in decibels (dB), with minimal attenuation for frequencies near 0 Hz, allowing these signals to pass unaffected. The red dashed line marks the cutoff frequency at 20 Hz, where the signal's magnitude starts to drop significantly, typically by -3 dB, indicating the transition to attenuation. Beyond this cutoff, higher frequencies are progressively suppressed, with greater attenuation indicated by more negative dB values. This filter is useful for removing high-frequency noise or undesired components, retaining only low-frequency signals for applications like smoothing data or preparing signals for further time-domain analysis. The steepness of the attenuation curve depends on the filter's order, with a steeper slope indicating a more selective filter.

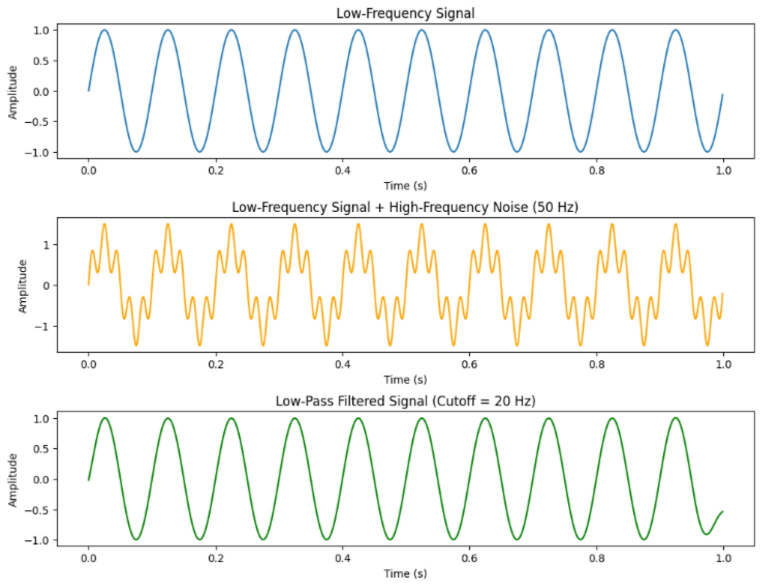


Figure 1B (a) – Effect of low-pass filter on a signal contaminated with high-frequency noise.

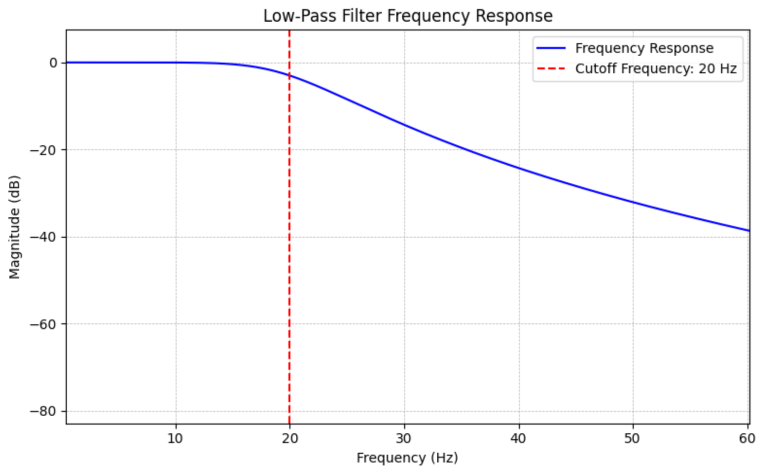


Figure 1B (b) – A 4th-order Butterworth low-pass filter

Figure 1C (a) demonstrates the application of a high-pass filter to isolate a high-frequency signal from low-frequency noise. The first subplot shows the original high-frequency signal, which is the desired data. The second subplot combines the high-frequency signal with low-frequency noise, creating a scenario common in real-world signal acquisition. The third subplot illustrates the result of applying a high-pass filter with a cutoff frequency of 30 Hz. The filter successfully removes the low-frequency noise while retaining the high-frequency signal, showcasing the effectiveness of high-pass filtering in applications requiring the preservation of rapid signal variations while eliminating slower, unwanted fluctuations.

The plot in Figure 1C (b) depicts the frequency response of a 4-th order high-pass filter, which allows signals with frequencies above the cutoff frequency to pass while attenuating lower frequencies. The blue curve represents the filter's magnitude response in decibels (dB), starting with significant attenuation at low frequencies. The red dashed line at 30 Hz marks the cutoff frequency, where the transition begins, typically defined by the -3 dB point. Beyond this frequency, the response levels off near 0 dB, indicating that higher frequencies are transmitted with minimal attenuation. This characteristic is useful for applications requiring the removal of low-frequency components, such as noise or drift, while preserving higher-frequency details in the signal. The steepness of the curve near the cutoff reflects the filter's design and order, with steeper slopes indicating greater selectivity in isolating higher frequencies.

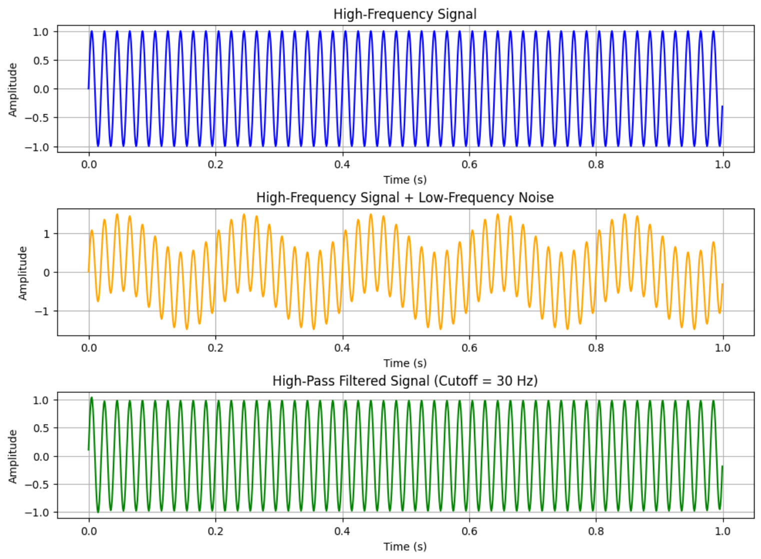


Figure 1C (a) – Effects of high-pass filter on a signal contaminated with low-frequency noise.

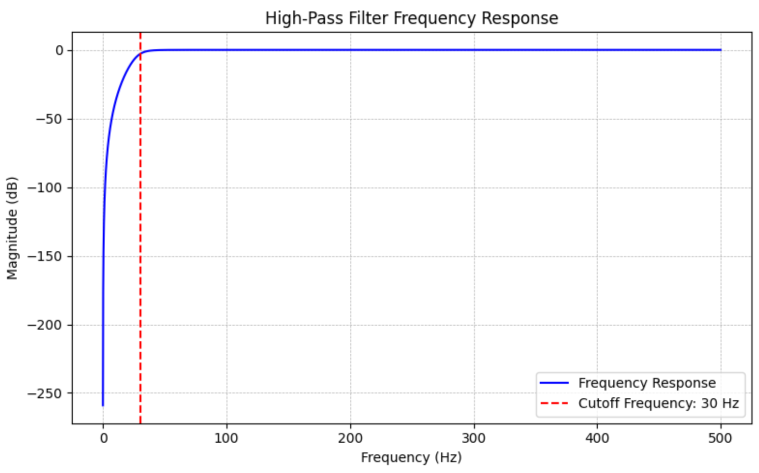


Figure 1C (b) – A 4th-order Butterworth high-pass filter

Band-pass filters are commonly used in communication systems to isolate specific frequency bands, while band-stop filters are applied to eliminate narrow bands of interference, such as power line noise at 50 or 60 Hz.

Figure 1D (a) demonstrates the effectiveness of a band-pass filter in isolating a desired frequency range from a noisy signal. The first subplot shows the original clean signal with a frequency of 10 Hz. The second subplot represents the signal contaminated with broadband noise, simulating real-world conditions where desired signals are often masked by unwanted noise. The third subplot illustrates the application of a bandpass filter with a frequency range of 8-12 Hz. The filter successfully retains the 10 Hz component while attenuating both lower and higher frequency noise, highlighting its utility in extracting specific frequency components for further analysis. This is particularly useful in applications like biomedical signal processing and communication systems where precise frequency isolation is required.

The plot in Figure 1D (b) illustrates the frequency response of a 4-th order band-pass filter designed to pass signals within a specific frequency range while attenuating those outside this range. The blue curve represents the filter's magnitude response in decibels (dB). The green dashed line at 8 Hz marks the lower cutoff frequency, and the red dashed line at 12 Hz marks the upper cutoff frequency, defining the passband of the filter. Within this range, the response remains near 0 dB, indicating minimal attenuation, while frequencies outside the passband are significantly attenuated, as seen by the steep drop-off in the response. This type of filter is commonly used to isolate frequency components within a specific band, such as in communication systems or signal analysis, where it is essential to focus on a narrow range of frequencies. The steepness of the transition at the cutoff frequencies depends on the filter design and order.

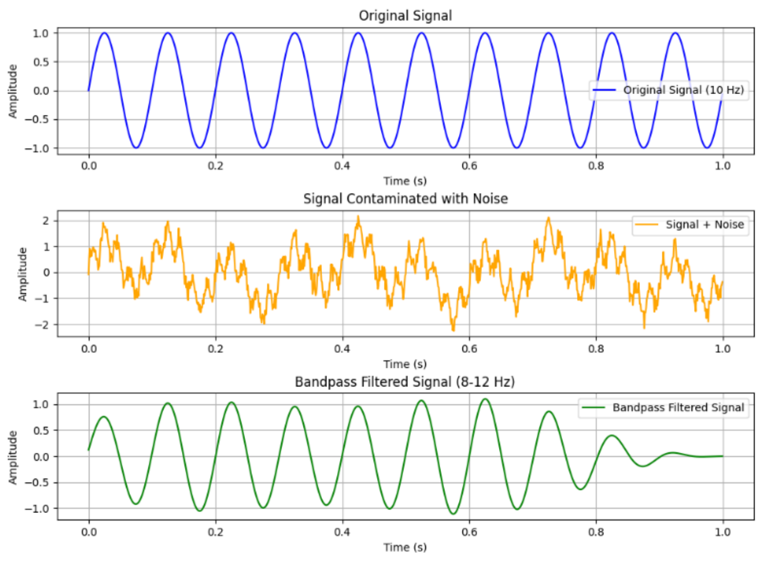


Figure 1D (a) – Effects of band-pass filter on a signal contaminated with broadband noise.

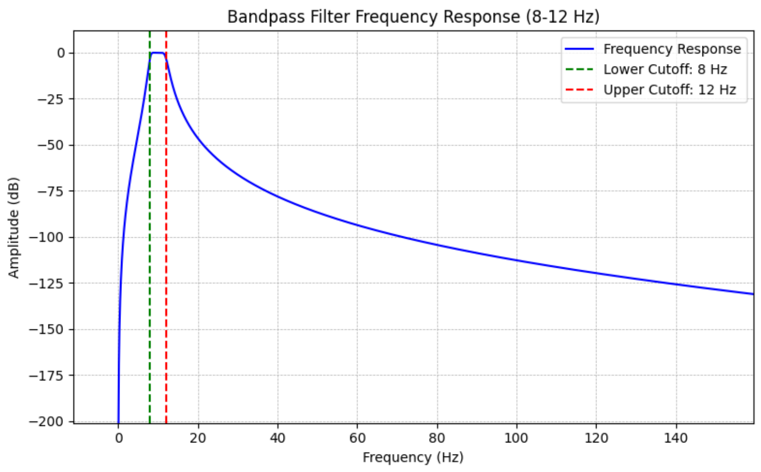


Figure 1D (b) – A 4th-order Butterworth band-pass filter

Figure 1E (a) illustrates the application of a band-stop filter to remove low-frequency noise from a high-frequency signal. The first subplot shows the original high-frequency signal at 30 Hz. The second subplot adds low-frequency noise at 5 Hz to simulate real-world conditions where a desired signal may be corrupted by low-frequency disturbances. In the third subplot, a band-stop filter is applied to attenuate the 5 Hz noise while preserving the 30 Hz high-frequency signal. The filtered signal demonstrates the effectiveness of the band-stop filter in selectively removing unwanted frequency components without significantly affecting the desired signal, which is particularly useful in applications such as audio processing and communication systems.

The plot in Figure 1E (b) depicts the frequency response of a 4-th order band-stop filter, which is designed to attenuate frequencies within a specific range while allowing frequencies outside that range to pass through with minimal attenuation. The blue curve represents the magnitude response of the filter in decibels (dB). The red dashed line at 4 Hz marks the lower cutoff frequency, and the green dashed line at 6 Hz marks the upper cutoff frequency, defining the stopband of the filter. Within this narrow band, the filter shows significant attenuation, as indicated by the dip in the magnitude response. Frequencies below 4 Hz and above 6 Hz experience negligible attenuation, maintaining their original amplitude. This type of filter is commonly used to suppress unwanted narrowband noise or interference, such as power line hum, while preserving the integrity of signals outside the stopband. The sharp transitions around the cutoff frequencies indicate the precision of the filter's design.

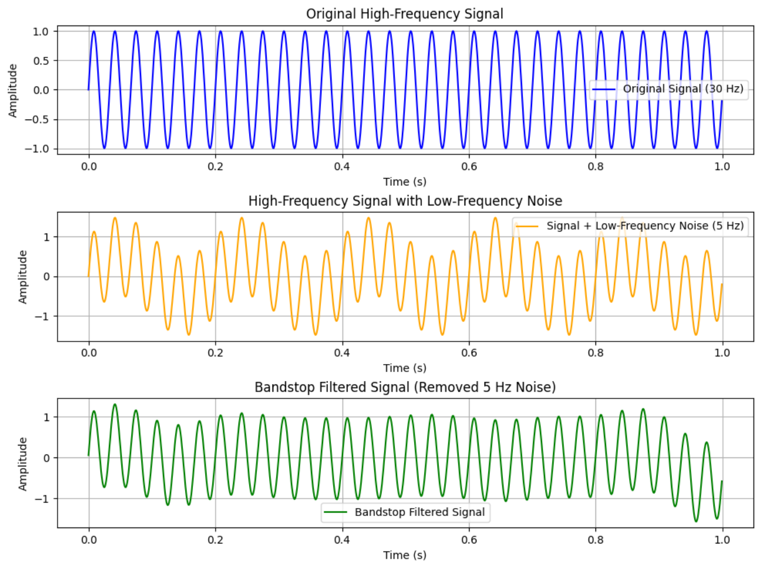


Figure 1E (a) – Effects of band-stop filter on a signal contaminated with a 5 Hz noise.

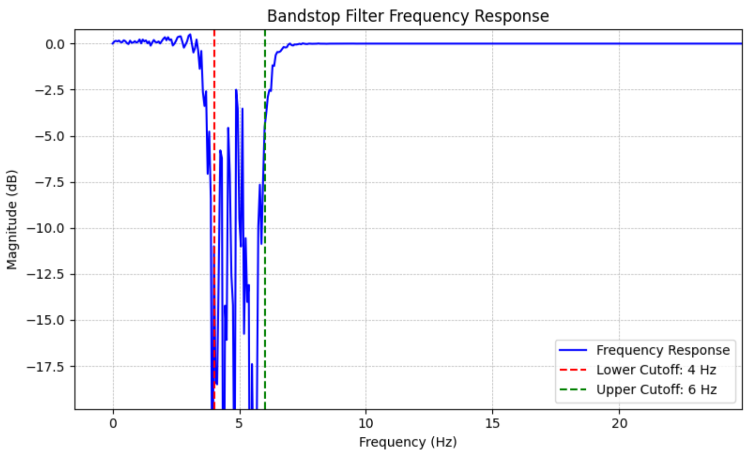


Figure 1E (b) – A 4th-order Butterworth band-stop filter

In real-time applications, signal pre-processing and filtering are critical for optimizing the performance of signal processing systems. In audio processing, for example, these techniques remove background noise and enhance speech clarity, making the output more intelligible. In communication systems, filtering ensures the integrity of transmitted signals by rejecting interference and noise. In biomedical applications, such as electroencephalography (EEG) or ECG analysis, preprocessing and filtering enable the extraction of clinically significant features from noisy physiological signals.

Advanced methods, such as adaptive filtering, are also gaining traction in modern systems. Adaptive filters adjust their parameters dynamically based on the input signal's characteristics, making them suitable for environments with time-varying noise. These approaches, combined with machine learning and artificial intelligence, are further pushing the boundaries of signal pre-processing and filtering, enabling real-time signal enhancement and robust feature extraction in complex scenarios.

2. Digital Signal Pre-Processing

2.1 Normalization

Signal normalization is a preprocessing technique used to scale data to a specific range or adjust its distribution to improve the performance and accuracy of subsequent signal analysis or machine learning tasks. In the context of signal processing, normalization is particularly important for handling signals with varying amplitudes or units, ensuring that all signals are on a common scale. This process is essential when combining multiple signals or when using algorithms sensitive to input scale, such as neural networks or frequency-domain transformations.

The simplest form of signal normalization involves scaling the signal to a range, such as [0, 1] or [-1, 1]. This can be achieved by dividing the signal by its maximum absolute value or by subtracting the minimum value and dividing it by the range. For example, for a signal $\boldsymbol{x(t)}$, min-max normalization can be expressed as:

$$\boldsymbol{x_{normalized}(t)} = \frac{\boldsymbol{x(t) - \min(x)}}{\boldsymbol{\max(x) - \min(x)}}$$

This approach is particularly effective when working with bounded signals, as it preserves the relative structure of the data while constraining it within the desired range.

In cases where signals have varying units or magnitudes, z-score normalization is often employed. This technique standardizes the signal by subtracting its mean and dividing by its standard deviation:

$$\boldsymbol{x_{normalized}(t)} = \frac{\boldsymbol{x(t) - \mu}}{\boldsymbol{\sigma}}$$

Where $\boldsymbol{\mu}$ is the mean and $\boldsymbol{\sigma}$ is the standard deviation of the signal. Z-score normalization ensures that the normalized signal has a mean of 0 and a standard deviation of 1, making it suitable for algorithms that assume data is centered and scaled.

Normalization is also critical in real-time systems, where signals must be consistently scaled for reliable processing. For instance, in audio signal processing, normalization ensures consistent loudness levels, while in biomedical signal processing, it allows for meaningful comparisons across signals acquired from different sensors or individuals. In machine learning applications, normalized signals help avoid numerical instability and bias in algorithms.

2.2 Scaling

Signal scaling is a process of adjusting the amplitude or range of a signal to suit specific requirements. This process involves transforming a signal so that its values fall within a desired range, either for compatibility with a system's input constraints, data normalization, or to emphasize or reduce certain features. Signal scaling is an essential step in many signal processing applications, such as audio processing, image processing, and data analysis, where the dynamic range of a signal needs to be adjusted to improve performance or compatibility with hardware or algorithms.

Signal scaling plays a crucial role in multiple domains. In digital systems, signals are often scaled to fit within the resolution of an Analog-to-Digital Converter (ADC) or a Digital-to-Analog Converter (DAC). For instance, an ADC with a 12-bit resolution can handle signals in a specific voltage range (e.g., 0 to 3.3V). Scaling ensures the signal fully utilizes the available resolution without exceeding the ADC's limits. Similarly, in machine learning and data analysis, scaling helps normalize data, making it easier for algorithms to converge during training or perform computations efficiently.

In audio processing, scaling is used to adjust the amplitude of sound waves to ensure consistent playback levels. It is also used in communication systems to match the signal power to the channel's dynamic range, optimizing signal-to-noise ratio and reducing distortion.

There are 4 common methods used for signal scaling:

1. **Min-Max Scaling**: This method adjusts a signal, so its values lie within a specified range, often [0, 1] or [-1, 1]. The formula for min-max scaling is:

$$\boldsymbol{x_{scaled}(t)} = \frac{\boldsymbol{x(t) - \min(x)}}{\boldsymbol{\max(x) - \min(x)}}$$

   Min-max scaling is widely used for preparing signals for machine learning algorithms or systems with bounded inputs.

2. **Amplitude Normalization**: In this method, the signal is scaled based on its maximum absolute value, often referred to as *peak scaling*. For example:

$$\boldsymbol{x_{scaled}(t)} = \frac{\boldsymbol{x(t)}}{\boldsymbol{\|\max(x)\|}}$$

   This ensures the scaled signal's amplitude does not exceed 1, making it suitable for systems with amplitude constraints, such as audio equipment or digital filters.

3. **Mean and Variance Scaling**: This technique involves centering the signal around zero mean and scaling by its standard deviation, typically used in statistical analysis:

$$\boldsymbol{x_{scaled}(t)} = \frac{\boldsymbol{x(t) - \mu}}{\boldsymbol{\sigma}}$$

   Where $\boldsymbol{\mu}$ is the mean and $\boldsymbol{\sigma}$ is the standard deviation. This method standardizes the signal, making it suitable for applications like Principal Component Analysis (PCA).

4. **Logarithmic Scaling**: This method applies a logarithmic transformation to compress dynamic ranges, especially useful for signals with large variations in amplitude, such as decibel scaling in audio processing:

$$\boldsymbol{x_{scaled}(t)} = \boldsymbol{\log(x(t))}$$

Scaling offers several benefits, including:

- Improved compatibility with system input ranges.
- Better numerical stability in algorithms, especially for machine learning.
- Enhanced interpretability of signals by reducing extreme variations.

However, improper scaling can introduce artifacts or distortions in the signal. For instance, scaling may amplify noise if not applied carefully, or it might reduce important signal features if the scaling parameters are incorrectly chosen. Therefore, it is crucial to select the appropriate scaling method based on the application and characteristics of the signal.

2.3 Smoothing

Signal smoothing is a fundamental technique in signal processing, used to reduce noise and fluctuations in data while retaining its important underlying trends and features. This process is particularly valuable in real-world scenarios where signals are often contaminated with noise due to environmental factors, sensor limitations, or other external interferences. The primary goal of smoothing is to make the signal more interpretable and suitable for further analysis, whether it's for visualization, feature extraction, or as a preprocessing step in advanced data processing workflows.

In many applications, raw signals can be challenging to interpret due to high levels of noise that mask the true features of the data. Smoothing helps in separating meaningful patterns from random variations, enabling better insights. For example, in biomedical engineering, smoothing techniques are applied to ECG or EEG signals to filter out noise caused by patient movement or electrical interference, improving the accuracy of diagnosis. Similarly, in industrial monitoring, smoothed sensor data can help identify gradual trends, such as wear and tear in machinery, that would otherwise be obscured by measurement noise.

There are several methods available for signal smoothing:

1. **Moving Average**: This is one of the simplest and most widely used methods. It calculates the average value of a signal over a specified window size, effectively reducing short-term fluctuations while maintaining the overall trend. A variant, the weighted moving average, assigns more importance to recent data points, making it more sensitive to recent changes.

   $$y[i] = \frac{1}{N} \sum_{j=1}^{i+N-1} x[j]$$

   Where:

   - $y[i]$: Smoothed signal
   - N: Window size
   - $x[j]$: Original signal

2. **Exponential Moving Average (EMA)**: This method uses exponentially decreasing weights for older data points, allowing it to react faster to changes in the signal compared to the simple moving average.
3. **Savitzky-Golay Filter**: This technique fits a polynomial function to a subset of the signal, preserving features like peaks and edges better than a moving average.
4. **Low-Pass Filtering**: By removing high-frequency components from a signal, low-pass filters effectively reduce noise while retaining the lower-frequency content, such as trends or slow-changing features.
5. **Median Filtering**: This method replaces each data point with the median value of a surrounding window. It is particularly effective at removing impulsive noise or outliers while preserving edges in the signal.

While signal smoothing is highly effective, it comes with trade-offs. Excessive smoothing can result in the loss of critical features, such as sharp transitions or peaks, which may contain valuable information. Choosing the right smoothing parameter, such as the window size in a moving average or the cutoff frequency in a filter, is crucial to achieving a balance between noise reduction and feature preservation. Additionally, the computational cost of certain advanced smoothing techniques can be a limitation in resource-constrained environments.

2.4 Transformation

Signal transformation refers to the process of converting a signal from one representation or domain to another to facilitate analysis, processing, or application-specific tasks. This concept is central to signal processing and is widely applied in fields like communications, audio processing, image analysis, and control systems. Transformations allow us to extract specific features, isolate relevant components, or simplify complex signals for further manipulation.

One common example of signal transformation is the **Fourier Transform**, which converts a time-domain signal into its frequency-domain representation. This transformation is particularly useful for analyzing periodic signals or identifying frequency components in noisy data. For instance, in audio processing, the Fourier Transform helps identify dominant frequencies in a sound signal, enabling applications like equalization, filtering, and noise reduction. Similarly, in vibration analysis, it can reveal specific frequency components indicative of machinery defects.

Another widely used transformation is the **Laplace Transform**, which extends the Fourier Transform's utility by considering signals in the complex frequency domain. This is particularly beneficial in control systems and circuit analysis, where it simplifies the representation of dynamic systems and aids in solving differential equations. Likewise, **Wavelet Transform** is increasingly popular for analyzing non-stationary signals, as it provides both time and frequency resolution, making it suitable for applications like image compression and biomedical signal processing.

Signal transformation is not limited to domain conversions; it also includes operations that modify the signal's structure or characteristics. For example, **logarithmic and exponential transformations** are used to compress dynamic ranges or emphasize certain signal features. In machine learning and data analysis, transformations like **normalization** and **standardization** adjust signal scales to improve model training and accuracy.

The importance of signal transformation lies in its ability to uncover hidden patterns, improve signal interpretability, and prepare data for specific tasks. For instance, transforming raw signals into spectrograms using **Short-Time Fourier Transform (STFT)** enables detailed time-frequency analysis, crucial in speech and music analysis. Additionally, transformations help in reducing noise, detecting edges in images, or extracting features in multidimensional data, broadening their applications in modern technologies.

3. Digital Signal Filtering

3.1 FIR and IIR Filters

Digital filters are fundamental tools in modern signal processing, designed to manipulate discrete-time signals by attenuating, enhancing, or isolating specific frequency components. Unlike their analog counterparts, digital filters operate on sampled data and are implemented using algorithms executed on digital platforms such as microcontrollers, digital signal processors, or general-purpose computers. These filters find applications across a wide range of fields, including audio processing, biomedical engineering, telecommunications, and industrial monitoring. Their versatility and precision make them indispensable in extracting meaningful information from noisy or distorted signals.

Digital filters are broadly classified into two main categories: **Finite Impulse Response (FIR)** filters and **Infinite Impulse Response (IIR) filters**. FIR filters rely solely on current and past input samples to compute the output, ensuring inherent stability and a linear phase response. These attributes make FIR filters ideal for applications where phase preservation is critical, such as audio equalization and image processing. FIR filter can be represented in the following discreet equation:

$$\boldsymbol{y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]}$$

Where $\boldsymbol{y[n]}$ is the output, $\boldsymbol{x[n-k]}$ are past inputs, and $\boldsymbol{b_k}$ are filter coefficients. FIR filter has finite number coefficients and depends only on current and past input values. **FIR filter is inherently stable** and does not have feedback loop, making it easy to design.

IIR filters, on the other hand, use both input and past output values, introducing feedback into the system. This allows IIR filters to achieve sharper frequency transitions with fewer coefficients, making them computationally efficient. However, care must be taken to ensure stability and phase distortion when designing IIR filters. This is the general mathematical equation for an IIR filter:

$$\boldsymbol{y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k] - \sum_{j=1}^{M} a_j \cdot y[n-j]}$$

Where:

- $\boldsymbol{y[n]}$: The current output of the filter at discrete time $\boldsymbol{n}$
- $\boldsymbol{x[n-k]}$: The current $(\boldsymbol{n})$ and past $(\boldsymbol{n-k})$ input samples

- $\boldsymbol{y}[\boldsymbol{n}-\boldsymbol{j}]$: The pass outputs of the filter (feedback)
- $\boldsymbol{b_k}$: The feedforward coefficients, determining how input samples affect the output
- $\boldsymbol{a_j}$: The feedback coefficients, determining how past outputs affect the output
- $\boldsymbol{N}$: The number of feedforward terms
- $\boldsymbol{M}$: The number of feedback terms

The design of digital filters involves specifying parameters such as the filter type (low-pass, high-pass, band-pass, or band-stop), cutoff frequencies, and filter order. Tools such as MATLAB, SciPy in Python, and various filter design libraries facilitate this process, generating coefficients that define the filter's frequency response. For instance, in audio processing, a low-pass filter might be used to remove high-frequency noise from a recording, while a bandpass filter can isolate specific frequency ranges in a sound signal for musical effects or speech analysis.

Despite their numerous advantages, digital filters face challenges such as computational overhead, especially in real-time applications. The finite resolution of digital systems can introduce quantization errors, and the choice of filter parameters significantly impacts performance. Nonetheless, advancements in hardware and algorithms continue to expand the capabilities of digital filters, enabling increasingly sophisticated applications in signal processing.

3.2 Digital Filter Design Example: Implementing a Digital Low-Pass FIR Filter

A low-pass FIR filter is to be designed using window method with the following specification:

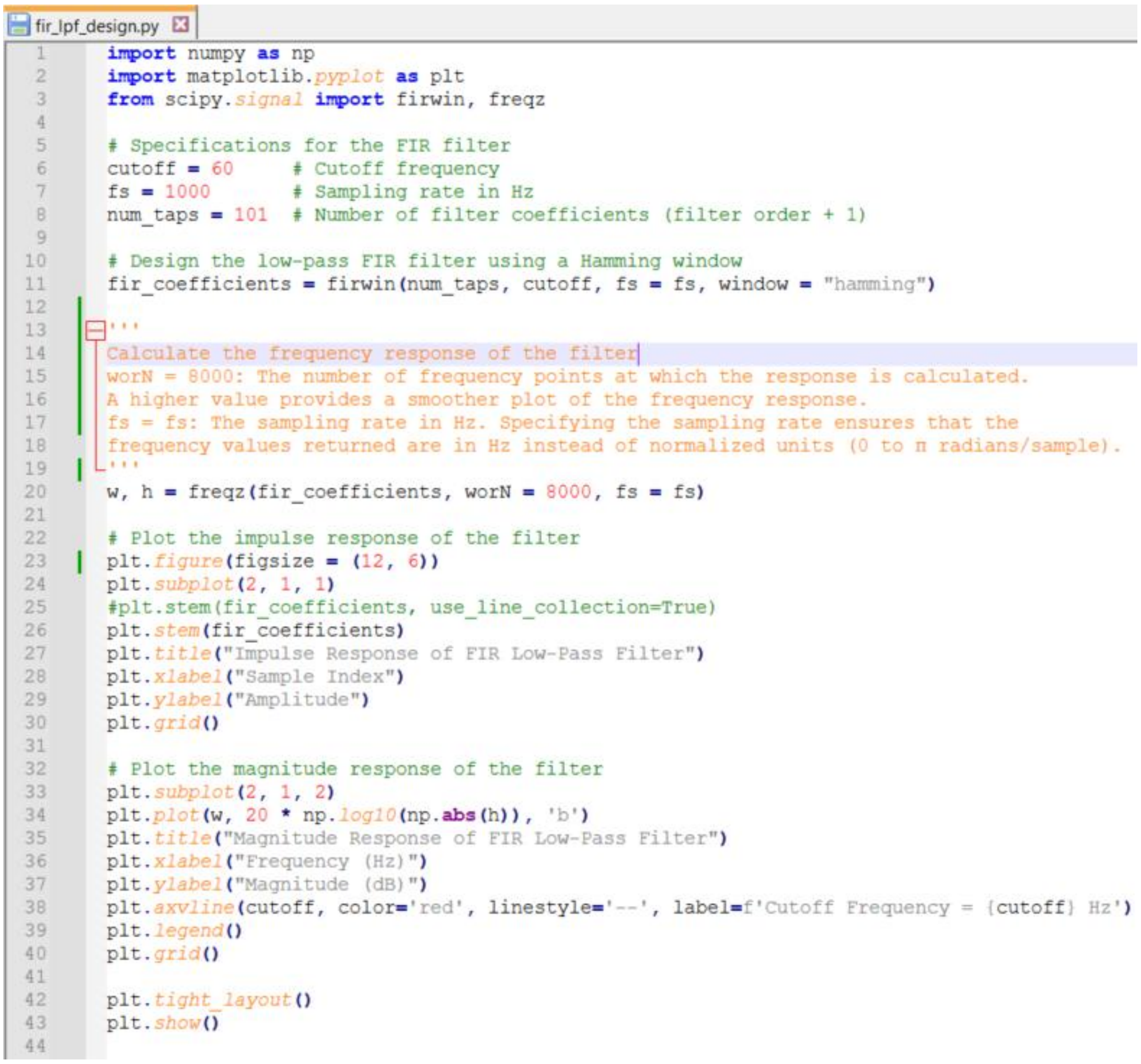
- Cutoff frequency = 60 Hz (or 0.12 if normalized to Nyquist frequency)
- Sampling rate = 1 kHz
- Filter order = 101
- Window type: Hamming window

The *scipy.signal* module in Python's SciPy library is a powerful tool for designing, analyzing, and applying digital filters. It supports the creation of various filter types, such as low-pass, high-pass, band-pass, and band-stop, enabling versatile signal processing applications. Filters can be designed using methods like *butter* for Butterworth filters, *cheby1* for Chebyshev filters, and *firwin* for FIR filter design using windowing techniques.

In addition to filter design, the module includes utilities for filter analysis and application. For example, the *freqz* function is commonly used to compute and visualize a filter's frequency response, allowing users to examine its behavior in terms of magnitude and phase. Once a filter is designed, functions like *lfilter* (applies a filter in direct form) and *sosfilt* (applies a filter in second-order sections for numerical stability) can process digital signals effectively.

Listing 3A depicts the completed Python script for the required digital filter design. This script demonstrates the design and analysis of a low-pass FIR filter using the *SciPy* library. The filter is created with a cutoff frequency of 60 Hz, a sampling rate of 1000 Hz, and 101 filter taps, utilizing a Hamming window to shape the filter coefficients. The *freqz* function calculates the frequency response, providing a detailed view of the filter's behavior in terms of frequency magnitude and phase. The script plots (Figure 3A) the impulse response of the filter, showing the filter coefficients in the time domain, as well as the magnitude response in the frequency domain, highlighting the attenuation of frequencies above the 60 Hz cutoff. This visualization aids in understanding the filter's design and verifying its performance for signal processing tasks.

Figure 3B illustrates the application of the digital low-pass filter in processing a noisy signal. The filter effectively removes the unwanted 300 Hz noise component while retaining the desired signal frequencies within the passband. The noisy signal, which initially includes both the desired signal and high-frequency noise, is passed through the filter. The result is a cleaner signal with the high-frequency noise significantly attenuated, demonstrating the filter's ability to isolate the desired frequency range and improve signal clarity.



```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import firwin, freqz

# Specifications for the FIR filter
cutoff = 60     # Cutoff frequency
fs = 1000       # Sampling rate in Hz
num_taps = 101  # Number of filter coefficients (filter order + 1)

# Design the low-pass FIR filter using a Hamming window
fir_coefficients = firwin(num_taps, cutoff, fs = fs, window = "hamming")

'''
Calculate the frequency response of the filter
worN = 8000: The number of frequency points at which the response is calculated.
A higher value provides a smoother plot of the frequency response.
fs = fs: The sampling rate in Hz. Specifying the sampling rate ensures that the
frequency values returned are in Hz instead of normalized units (0 to π radians/sample).
'''
w, h = freqz(fir_coefficients, worN = 8000, fs = fs)

# Plot the impulse response of the filter
plt.figure(figsize = (12, 6))
plt.subplot(2, 1, 1)
#plt.stem(fir_coefficients, use_line_collection=True)
plt.stem(fir_coefficients)
plt.title("Impulse Response of FIR Low-Pass Filter")
plt.xlabel("Sample Index")
plt.ylabel("Amplitude")
plt.grid()

# Plot the magnitude response of the filter
plt.subplot(2, 1, 2)
plt.plot(w, 20 * np.log10(np.abs(h)), 'b')
plt.title("Magnitude Response of FIR Low-Pass Filter")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude (dB)")
plt.axvline(cutoff, color='red', linestyle='--', label=f'Cutoff Frequency = {cutoff} Hz')
plt.legend()
plt.grid()

plt.tight_layout()
plt.show()

```

Listing 3A - Digital low-pass FIR Filter design

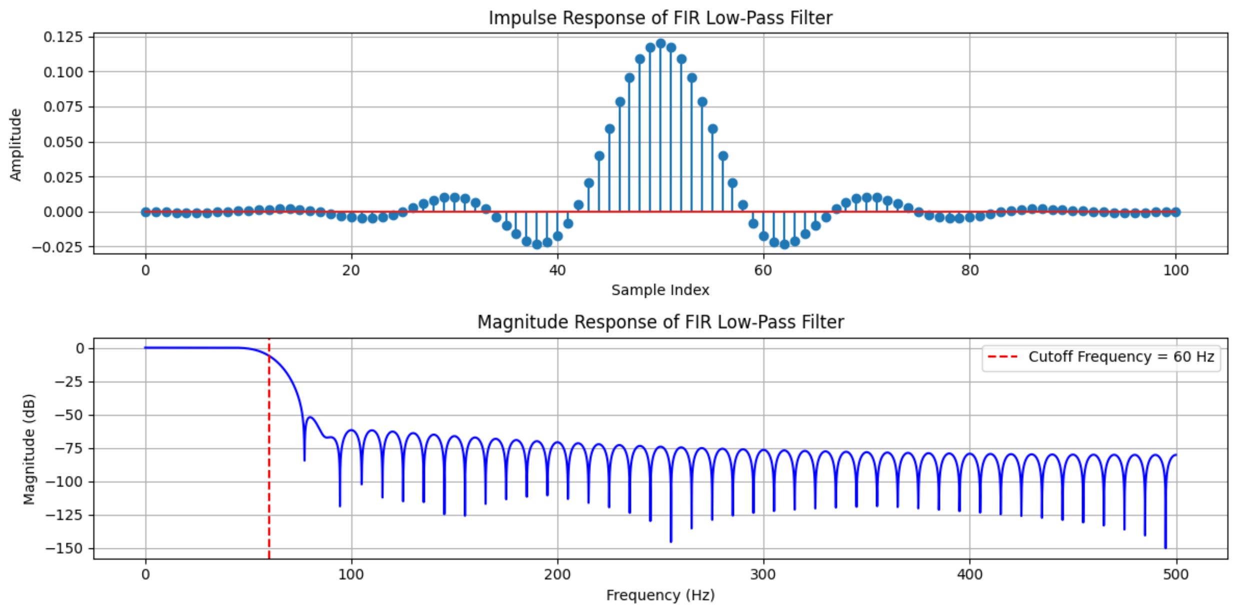


Figure 3A - Digital low-pass FIR Filter Implementation

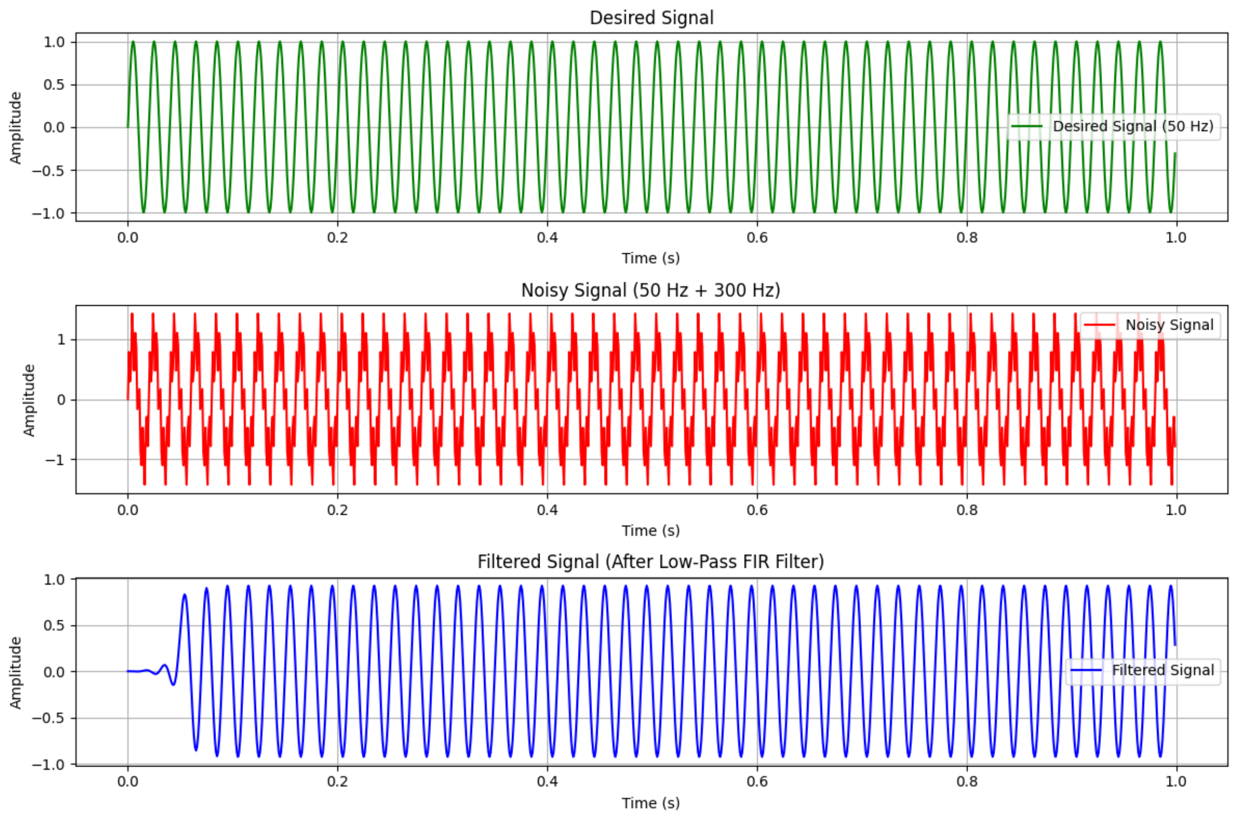


Figure 3B - Digital low-pass FIR Filter filtering out unwanted 300Hz noise

4. Applications of Signal Pre-Processing and Filtering in Arduino

4.1 Distance Sensing with Ultrasonic Sensor

The Grove - Ultrasonic Ranger can measure distances ranging from a few centimeters to several meters, making it suitable for both indoor and outdoor environments. However, raw measurements from the ultrasonic sensor can occasionally be noisy due to environmental interference or multi-path reflections. To address this, signal pre-processing techniques such as (but not limited to) **averaging, moving average filtering** or **median filtering** can be employed to smooth the readings and enhance accuracy.

A simple ultrasonic distance measurement system (covered in Lesson 2) can be assembled by connecting the Ultrasonic Ranger to a digital input/output port (e.g. D3) of Arduino MKR WiFi 1010 board via the MKR Connector Carrier Board, and the Grove RGB LCD to the Arduino MKR TWI (I2C) port.

The Arduino program triggers the ultrasonic sensor, captures the echo response, and calculates the distance. The distance value is then processed using signal pre-processing techniques before being displayed on the RGB LCD.

Several open-sourced signal pre-processing libraries available for Arduino:

1. Moving Average Filter

   https://github.com/sebnil/Moving-Avarage-Filter--Arduino-Library-/tree/master

2. Median Filter

   https://github.com/luisllamasbinaburo/Arduino-MedianFilter

A summary of possible sensor data pre-processing methods:

1. **Averaging** is the simplest method used to process distance measurements from the ultrasonic sensor. In this technique, the mean of a set of consecutive measurements is calculated to smooth out noise and fluctuations in the data. Averaging is computationally lightweight and straightforward to implement. However, it has its drawbacks: **outliers** can significantly skew the results, and the method can be **slow to respond to rapid changes** in the measured distance.

2. **Moving Average Filtering** builds upon simple averaging by maintaining a **sliding window** of the most recent measurements. As new data enters the window, the oldest data point is removed, ensuring that the filter dynamically adapts to the latest conditions. This approach effectively reduces noise while maintaining better responsiveness to changes in distance compared to simple averaging. It strikes a balance between signal smoothness and real-time performance. However, like averaging, it is less effective in handling significant outliers unless combined with additional techniques for outlier rejection.

3. **Median Filtering** is a more robust technique for processing distance data, particularly in environments prone to impulsive noise or outliers. Unlike averaging, which computes the mean, median filtering replaces each data point with the median value of a sliding window of measurements. This **non-linear filtering** method is highly effective at eliminating outliers, as the

median is not influenced by extreme values. Additionally, it preserves sharp transitions in the signal, such as sudden changes in distance, making it ideal for applications requiring precise measurements in dynamic conditions. However, **median filtering is computationally more intensive** than averaging or moving average filtering due to the need to sort the values in the window. It is best suited for noisy or unpredictable environments, such as outdoor distance sensing, where robustness against outliers is critical.

4.2 Sound Monitoring

The Sound Sensor module is an Electret Microphone connected to MAX9814 amplifier with Automatic Gain Control to be used for sound detection and monitoring applications.

In a typical hardware setup (covered in Lesson 2), the sound sensor is interfaced with Arduino Board's analog input pin (e.g. A2) to capture and process sound levels. Integrating digital FIR filtering enhances the capabilities of the sound sensor by improving the quality and precision of sound analysis and processing.

The Arduino program processes sound signals by implementing a **FIR filtering algorithm**. This algorithm is applied to the raw sound signals captured by the sound sensor, to remove noise and enhance the quality of the signals. By leveraging the FIR filter, the program effectively suppresses unwanted frequency components while retaining the desired features of the sound.

The noise-suppressed output from the FIR filter is particularly valuable for various applications. In **signal analytics**, it enables the extraction of meaningful patterns and trends from sound data, which can be used in tasks such as speech recognition, audio diagnostics, or environmental noise analysis. In **sound-related applications**, filtered signals enhance system performance, ensuring clarity and accuracy in use cases such as voice-activated controls, sound-level monitoring, and acoustic event detection.

FIR filter can be designed using online tools:

1. TFilter

   http://t-filter.engineerjs.com/

2. FIIIR!

   https://fiiir.com/

FIR filter library implementation in Arduino:

https://github.com/LeemanGeophysicalLLC/FIR_Filter_Arduino_Library

https://github.com/EmotiBit/EmotiBit_ArduinoFilters

FIR Filter Application Examples

One of the primary applications of FIR filtering with the sound sensor is **noise reduction**. In noisy environments, sound signals often contain unwanted background noise that can obscure important audio information. Low-pass or band-pass FIR filters can be used to suppress irrelevant noise and isolate desired frequency ranges. For instance, low-pass FIR filters are effective in removing high-frequency noise, while band-pass FIR filters can target specific ranges such as the human speech spectrum (300 Hz to 3 kHz). This ensures cleaner signals for applications like voice communication or sound monitoring.

In **speech processing**, FIR filters play a critical role in refining audio signals to enhance clarity and intelligibility. High-pass FIR filters can suppress low-frequency sounds like rumble, making voice signals clearer. Band-pass FIR filters focused on speech frequencies further improve signal fidelity, making the sound sensor suitable for voice-activated systems or speech recognition applications, even in noisy conditions.

In **environmental monitoring**, FIR filters ensure accurate sound-level measurements in public spaces or industrial environments. For instance, low-pass FIR filters can isolate general background noise levels for compliance with noise regulations, while band-stop FIR filters can exclude specific frequency ranges like machinery noise to focus on other relevant environmental sounds. Similarly, in **acoustic event detection**, FIR filters are used to isolate characteristic frequency ranges of events like clapping, door slams, or breaking glass, improving detection reliability for security or automation systems.

In **biomedical applications**, the combination of the sound sensor and FIR filtering offers precise sound processing for tasks such as auscultation, where body sounds like heartbeats or lung sounds are analyzed. By filtering out environmental noise and focusing on specific frequency ranges, the system improves diagnostic accuracy.

Other notable usages of FIR filtering are in the areas of **audio equalization** and **audio spectrum analysis**.

4.3 Inertial Navigation System (Advanced Content)

A **tri-axial accelerometer** is a sensor designed to measure acceleration along three orthogonal axes (X, Y, and Z). This data is fundamental in **inertial navigation systems (INS)**, which calculate an object's position, velocity, and orientation without relying on external references like GPS. Inertial navigation is critical for applications in aerospace, robotics, automotive systems, and mobile devices, especially in environments where GPS signals are unavailable or unreliable.

The operation of a tri-axial accelerometer in INS begins with the measurement of linear acceleration due to motion, gravity, or a combination of both. These raw acceleration values are processed through integration over time to determine velocity (single integration) and position (double integration). When combined with orientation data from gyroscopes or other sensors, this information allows INS to calculate the object’s trajectory with high precision.

To ensure accurate results, the accelerometer's data undergoes several processing steps. First, noise filtering techniques, such as **low-pass filters** or **Kalman filters**, are applied to smooth the data and eliminate high-frequency noise. Next, gravity compensation is performed by leveraging orientation data from gyroscopes or magnetometers to isolate motion-induced acceleration from the gravitational component. Following this, numerical integration methods, like the trapezoidal rule, are employed to convert acceleration data into velocity and position. However, since integration can lead to error accumulation over time, drift correction is crucial. This is often achieved through sensor fusion, combining accelerometer data with inputs from GPS or barometers.

Tri-axial accelerometers find applications in various domains. In **aerospace navigation**, INS ensures accurate navigation for aircraft and spacecraft in GPS-denied environments. In **robotics**, accelerometers aid in path planning and autonomous movement by providing real-time position and velocity data. In **automotive systems**, they are used for navigation, crash detection, and advanced driver-assistance systems (ADAS). Portable devices, such as smartphones and wearables, also leverage accelerometers for motion detection, step tracking, and user interaction.

INS Example: A Pitch and Roll Measurement System

A pitch and roll measurement system utilizes a tri-axial accelerometer to detect orientation by measuring tilt angles in both pitch and roll directions. The Grove - 3-Axis Digital Accelerometer (covered in Lesson 2) is connected to the TWI (I2C) port of the Arduino MKR WiFi 1010 via the MKR Connector Carrier Board to acquire real-time acceleration data. To provide a visual output, a Grove - RGB LCD module is also connected to the same TWI port, displaying the computed pitch and roll angles.

The accelerometer gathers raw acceleration data across the X, Y, and Z axes. This data is preprocessed using a **Kalman filter** to suppress noise and enhance accuracy, ensuring stable measurements. Once filtered, the data undergoes trigonometric computations to determine the pitch and roll angles relative to the horizontal plane. The Arduino MKR WiFi 1010 serves as the processing hub, efficiently performing these calculations and relaying the results to the RGB LCD module for visualization.

Kalman filter library implementation in Arduino:

https://github.com/denyssene/SimpleKalmanFilter

https://github.com/TKJElectronics/KalmanFilter

https://github.com/rfetick/Kalman

- The End -