Measurement Systems

3rd Lab Session

Professor: Dr. Amini

Amirhossein Ansari 810600050

Hana Shamsaei 810600097

Spring 1404

Department of Mechanical Engineering

University of Tehran

Contents

1 Introduction

In this lab session, we focused on generating and modifying analog signals using filters and simple electronic circuits. All three assignments involved building systems with an Arduino and analyzing their behavior using MATLAB. The main goal was to understand how filters affect signals in practical situations.

In the first assignment, we designed a system that generates a sinusoidal signal using PWM. The frequency of the sinusoid could be adjusted by turning a potentiometer and pressing a pushbutton. We had to select resistor and capacitor values so that the low-pass filter would allow a 175 Hz signal to pass with no more than 3 dB attenuation. The output signal was plotted in MATLAB to compare it with the ideal waveform.

In the second assignment, we generated a signal that was a combination of a low-frequency and a high-frequency sinusoid, both with the same amplitude. A low-pass filter was used to remove the high-frequency component and recover the low-frequency signal. We also calculated the amount of attenuation for both frequencies and explained the result.

The third assignment was about testing the frequency response of one of the filters from the previous tasks. We applied at least 10 different frequencies to the filter, especially around the cut-off frequency, and calculated the attenuation at each point. The measured results were compared with the theoretical response to understand how closely the filter matched the design.

2 Filters

2.1 What is a filter?

In signal processing, a *filter* is a circuit that changes a signal by letting some frequencies pass through while reducing or blocking others. Filters can be made using digital methods or analog components. In this lab session, we focus on **passive analog filters**, which are built using basic components like resistors and capacitors. In more complex cases, inductors can also be used.

2.2 Why do we need filters?

Filters are useful in many real-world systems because they help clean up signals or focus on the parts we care about. In mechanical and control systems, filters are often used to:

- remove unwanted high-frequency noise from PWM signals to create a smooth output,
- separate different vibration signals to help detect problems in machines,
- prevent aliasing before converting analog signals to digital, and
- clean and adjust signals from sensors (like thermocouples or strain gauges) so that they work properly with other devices in the system.

2.3 Types of Filters

Filters are usually grouped based on how they affect the frequency content of a signal. The most common types are:

- **Low-pass filter (LPF)** — allows low frequencies to pass and reduces higher ones. It is commonly used to smooth out signals.

- **High-pass filter (HPF)** — allows high frequencies to pass and reduces lower ones. It can remove offsets or highlight quick changes in a signal.
- **Band-pass filter (BPF)** — allows only a certain range of frequencies to pass and blocks both lower and higher ones. It is useful when we want to isolate a specific frequency band.
- **Band-stop filter (BSF) or Notch filter** — blocks a narrow range of frequencies while letting the rest pass. For example, it can remove 50/60 Hz noise from electrical signals.

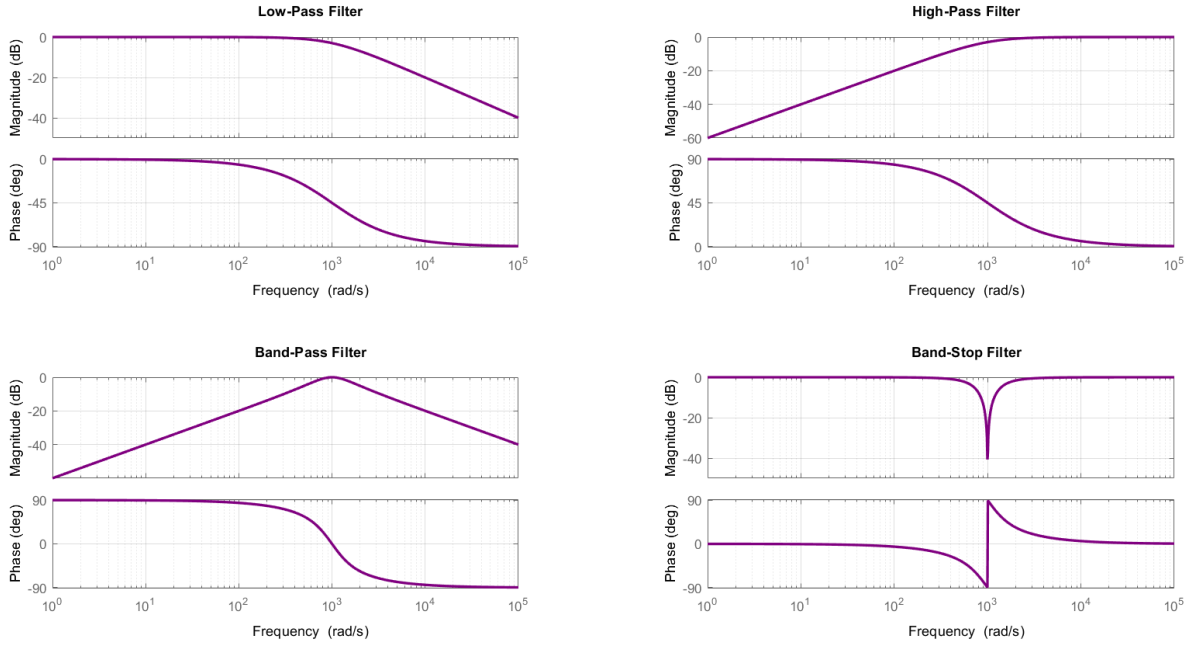

Figure 1: Ideal magnitude for first-order low-pass, high-pass, band-pass and band-stop filters.

The *order* of a filter tells us how many reactive components (like capacitors or inductors) are used in the circuit. Each extra component increases the filter's order and makes the signal drop faster after the cut-off frequency — usually by about 20 dB per decade for each order. It also causes a sharper change in the phase.

In this lab, we only work with **first-order RC filters**, which use just one resistor and one capacitor. Their transfer functions are:

$$H_{\mathrm{LPF}}(s) = \frac{1}{1+RCs}\,, \qquad H_{\mathrm{HPF}}(s) = \frac{RCs}{1+RCs}\,,$$

The cut-off frequency (also called the corner frequency) happens at:

$$f_c = \frac{1}{2\pi RC}$$

These basic filters are easy to build on a breadboard and are useful for analyzing how signals change over time (such as step or ramp responses).

In real circuits, resistors and capacitors do not behave exactly as ideal components. They have small internal resistances, inductance, and losses that can slightly affect the performance of a filter—especially

at high frequencies. These effects may cause a small shift in the cut-off frequency or reduce the sharpness of the filter, and can be observed during practical measurements like those in Assignment 3.

The filters used in this lab are passive, meaning they only use resistors and capacitors and do not require a power supply. Passive filters are simple, stable, and easy to build, which makes them suitable for many basic signal processing tasks.

In general, filters can also be active (using op-amps to boost performance) or digital (implemented in software). Sometimes, both analog and digital filters are used together in a system.

When designing a filter, the cut-off frequency is one of the most important factors. It depends on the values of the resistor and capacitor, and must be chosen to match the desired signal range. The time it takes for the filter to respond is also based on these values. For applications like smoothing PWM signals from an Arduino, the filter must be strong enough to remove high-frequency components while still responding fast enough to track changes in the signal.

3 Assignment 1

3.1 Procedure

To generate a sinusoidal signal with adjustable frequency and a smooth output waveform, we needed to design a low-pass filter. The goal of the filter was to remove high-frequency components from the PWM signal generated by the Arduino and let the desired sinusoidal shape pass through.

According to the assignment, the filter should allow a frequency of 175 Hz to pass with no more than 3 dB attenuation. In an RC low-pass filter, the frequency at which the signal's power drops by 3 dB is called the **cutoff frequency**, and it is defined by the following formula:

$$f_c = \frac{1}{2\pi RC}$$

The cutoff frequency formula comes from analyzing the transfer function of a basic RC low-pass filter. In this circuit, the output is taken across the capacitor, and the impedance of the capacitor is frequency-dependent:

$$Z_C = \frac{1}{j\omega C}$$

Using the voltage divider rule, the transfer function becomes:

$$H(j\omega) = \frac{V_{\text{out}}}{V_{\text{in}}} = \frac{1}{1 + j\omega RC}$$

The magnitude of the transfer function is:

$$|H(j\omega)| = \frac{1}{\sqrt{1 + (\omega RC)^2}}$$

The cutoff frequency f_c is defined as the frequency where the output drops by 3 dB, meaning the magnitude is $\frac{1}{\sqrt{2}}$. Solving:

$$\frac{1}{\sqrt{1 + (\omega_c RC)^2}} = \frac{1}{\sqrt{2}} \Rightarrow \omega_c RC = 1$$

$$\omega_c = \frac{1}{RC}, \quad \text{and} \quad f_c = \frac{\omega_c}{2\pi} = \frac{1}{2\pi RC}$$

We chose a capacitor value of:

$$C = 1\ \mu\text{F} = 1 \times 10^{-6}\ \text{F}$$

Then, using the cutoff frequency formula, we calculated the required resistance R for the filter:

$$R = \frac{1}{2\pi f_c C} = \frac{1}{2\pi \cdot 175 \cdot 1 \times 10^{-6}} \approx 909\ \Omega$$

Since 909 Ω is not a standard resistor value and precise tuning was needed, we used a potentiometer as a variable resistor. A potentiometer allows manual adjustment of resistance. We connected it in series

with the capacitor to form the RC filter, and used a multimeter to measure the resistance between the wiper and one terminal. We slowly adjusted the potentiometer until it showed a resistance close to 909 Ω.

Once the correct value was set, we fixed the position of the potentiometer and completed the filter.

3.2 Circuit Description

3.2.1 Components:

The circuit built for this assignment consists of the following main components:

- **Arduino Uno** – used to generate the PWM signal and read inputs from the potentiometer and pushbutton.
- **Two Potentiometers** – one is used to control the frequency of the output sine wave, and the other is used as a variable resistor in the RC low-pass filter.

  A **potentiometer** is a three-terminal variable resistor commonly used to adjust voltage levels in a circuit. It works as a voltage divider and is widely used for controlling analog signals such as brightness, volume, and—in our case—frequency input to the Arduino.

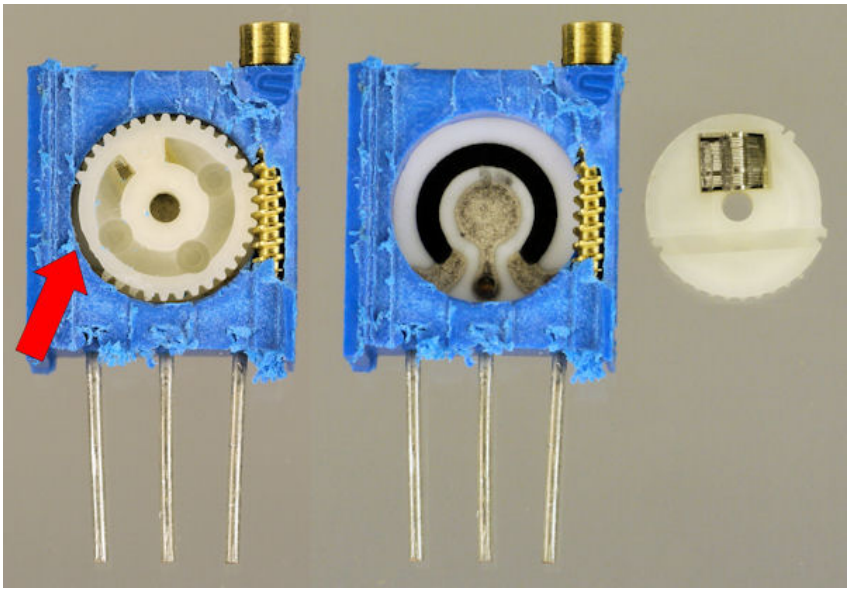

Figure 2: Internal structure of a rotary potentiometer

The internal structure of a potentiometer, shown in Fig. 2, consists of:

- **Resistive Track:** A semicircular strip made of resistive material (seen in black) that connects the two outer terminals.
- **Wiper:** A movable metal contact (middle pin) that slides along the resistive track as the knob turns. It determines the output voltage by dividing the total resistance.
- **Rotating Shaft and Gear:** Mechanically moves the wiper across the resistive track.

When a voltage is applied across the two end terminals of the potentiometer, the voltage at the middle terminal (wiper) depends on the position of the wiper along the resistive track. This allows continuous adjustment of voltage from 0 V (one end) to the full supply voltage (the other end). In our experiment, the potentiometer was used to control either the analog frequency input (via analog pin A1) or the resistance in the RC low-pass filter.

- **Pushbutton** – used to confirm and apply the updated frequency from the potentiometer.
- **Capacitor (1 µF)** – forms an RC low-pass filter together with the potentiometer to smooth the PWM signal.
- **Breadboard and Jumper Wires** – used for making all the necessary connections between the Arduino and components.

3.2.2 Circuit Logic and Connections:

- **Potentiometer for frequency control:** The first potentiometer, located on the left side of the breadboard, acts as a user-controlled input to set the frequency of the output sinusoidal signal. Its two outer pins are connected to the Arduino's 5 V and GND rails, while the middle (wiper) pin is connected to analog input pin A1. As the knob is turned, the voltage at the wiper changes linearly between 0 V and 5 V. This voltage is read by the Arduino and mapped to a frequency value between 0 and 100 Hz. However, this value is not updated continuously—instead, it is read only when the pushbutton is pressed. This design choice prevents the frequency from fluctuating unintentionally due to small mechanical movements or noise in the potentiometer.
- **Pushbutton for confirming frequency selection:** The pushbutton is connected between digital pin D2 and GND. Internally, it is configured with a pull-up resistor using `INPUT_PULLUP`, which keeps the default logic level HIGH. When the button is pressed, the pin is pulled to LOW and then returns HIGH upon release. An interrupt is configured on the rising edge of this signal, so the Arduino detects when the user finishes pressing the button. At that point, the new analog value from A1 is read, converted into a frequency value, and applied to generate the updated sine wave.
- **Potentiometer used as variable resistor (RC filter):** The second potentiometer, located on the right side of the breadboard, is used as a variable resistor in the low-pass RC filter. To achieve the desired cutoff frequency (175 Hz), we calculated the required resistance using a 1 µF capacitor. The potentiometer was adjusted manually using a multimeter until the resistance between its wiper and one terminal reached approximately 909 Ω. Once adjusted, the potentiometer was fixed in place and used as a static resistor. One end of this potentiometer is connected to the Arduino's PWM output (pin D11), while the wiper is connected to the capacitor.
- **Capacitor and RC filter output:** The 1 µF capacitor is connected between the wiper of the second potentiometer and ground. This forms a first-order RC low-pass filter, which smooths the PWM signal from the Arduino and converts it into an analog voltage waveform resembling a sine wave. The output voltage across the capacitor (the filtered signal) is then connected to analog input pin A0. This allows the Arduino to sample the waveform and send the data to MATLAB for plotting and comparison.

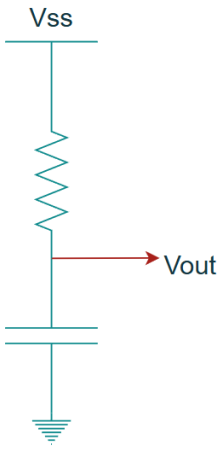

Figure 3: RC filter circuit

- **Serial communication with MATLAB:** In the main loop, the Arduino continuously samples the analog signal from A0 and records the current time using `micros()`. It then sends the timestamp, the filtered voltage (converted to volts), and the current frequency value over the serial port. MATLAB listens to this serial stream and uses it to plot the output waveform in real time. This setup allows us to visualize and analyze the waveform shape and how it changes with different frequency settings.

This circuit design allows precise user control over the output frequency using a potentiometer and a pushbutton for confirmation. The use of an RC low-pass filter ensures that the PWM output is transformed into a clean analog waveform. Together, these features provide a practical system for studying signal generation, filtering, and data acquisition in embedded systems.

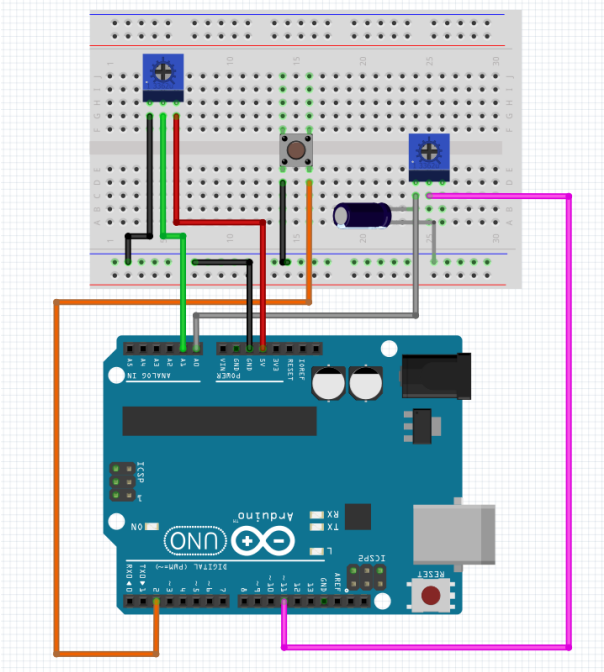

Figure 4: Fritzing of the first assignment

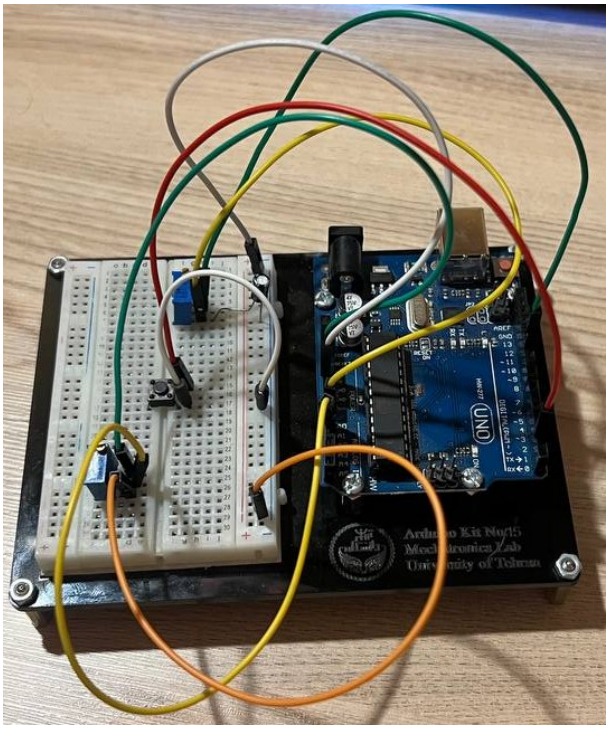

Figure 5: circuit of the first assignment

3.3 Results

Now the results for different frequencies in Arduino and MATLAB will be shown below. First the Arduino result (Serial Plotter) will be shown and then the MATLAB result will be shown :

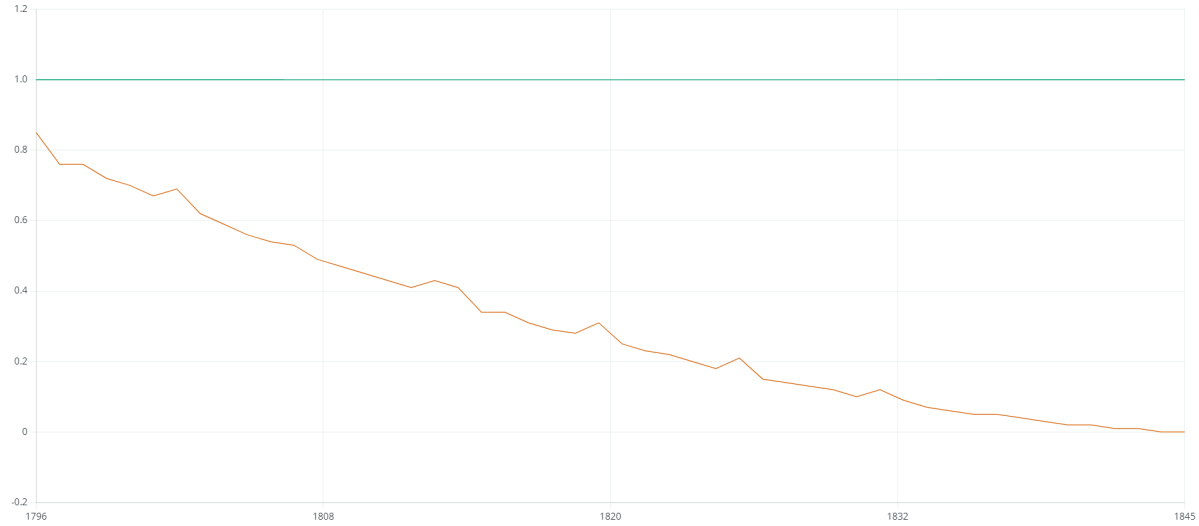

Figure 6: Arduino result for f = 1 Hz

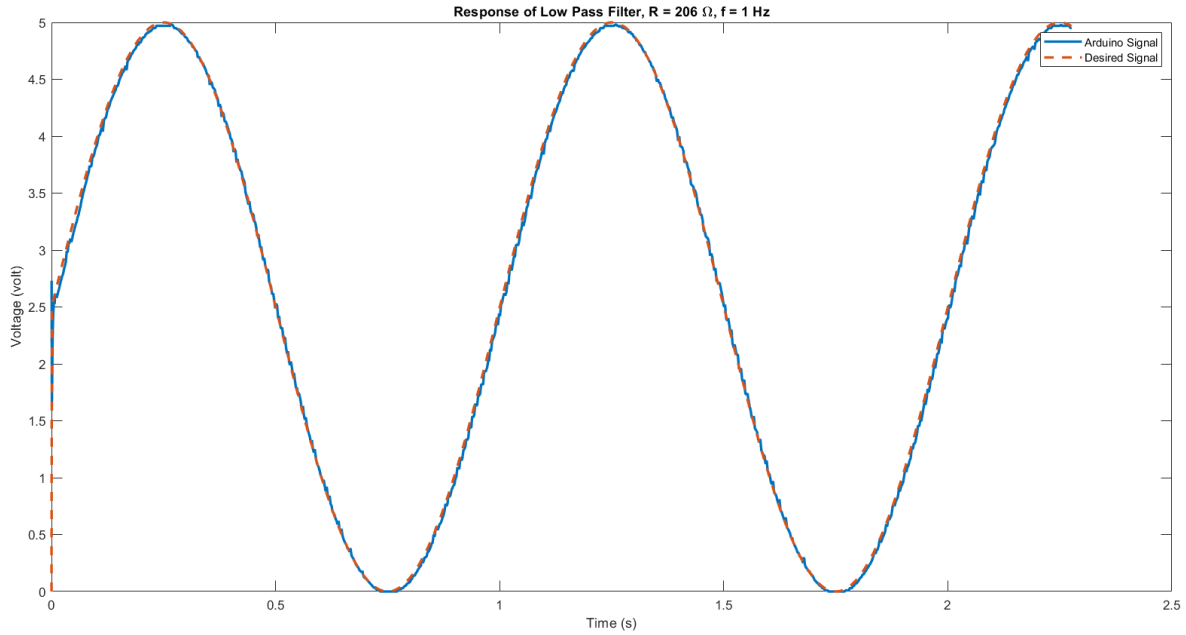

Figure 7: MATLAB result for f = 1 Hz

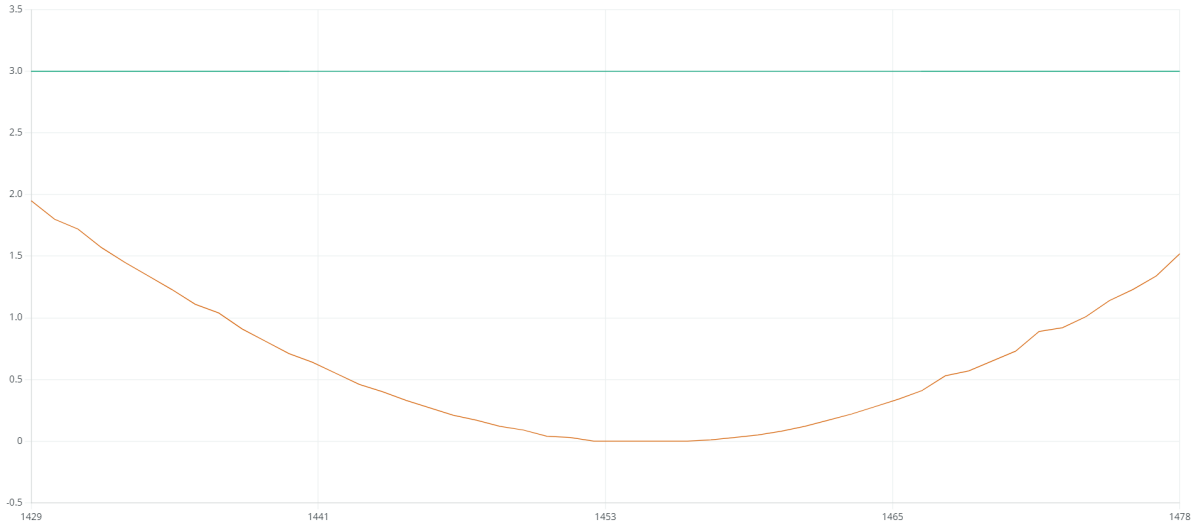

Figure 8: Arduino result for f = 3 Hz

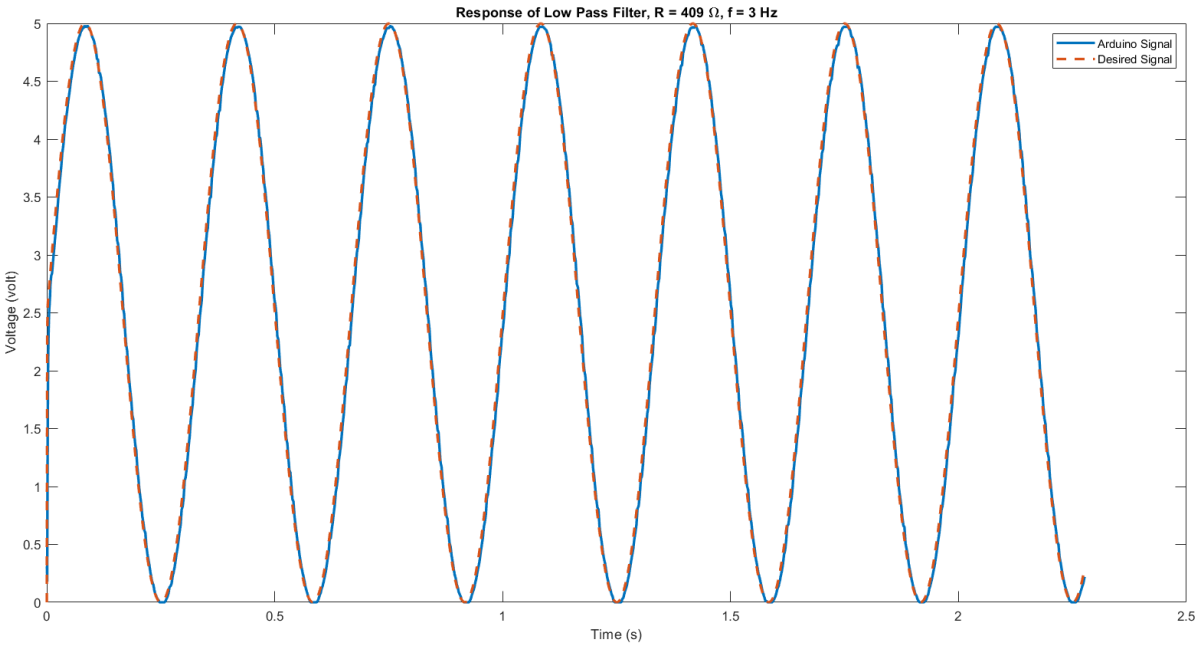

Figure 9: MATLAB result for f = 3 Hz

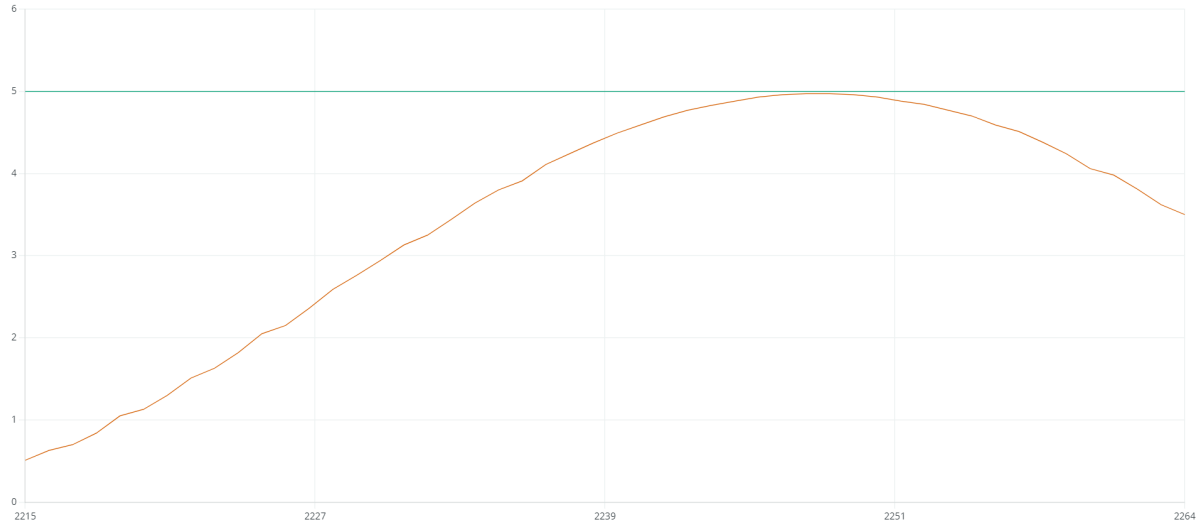

Figure 10: Arduino result for f = 5 Hz

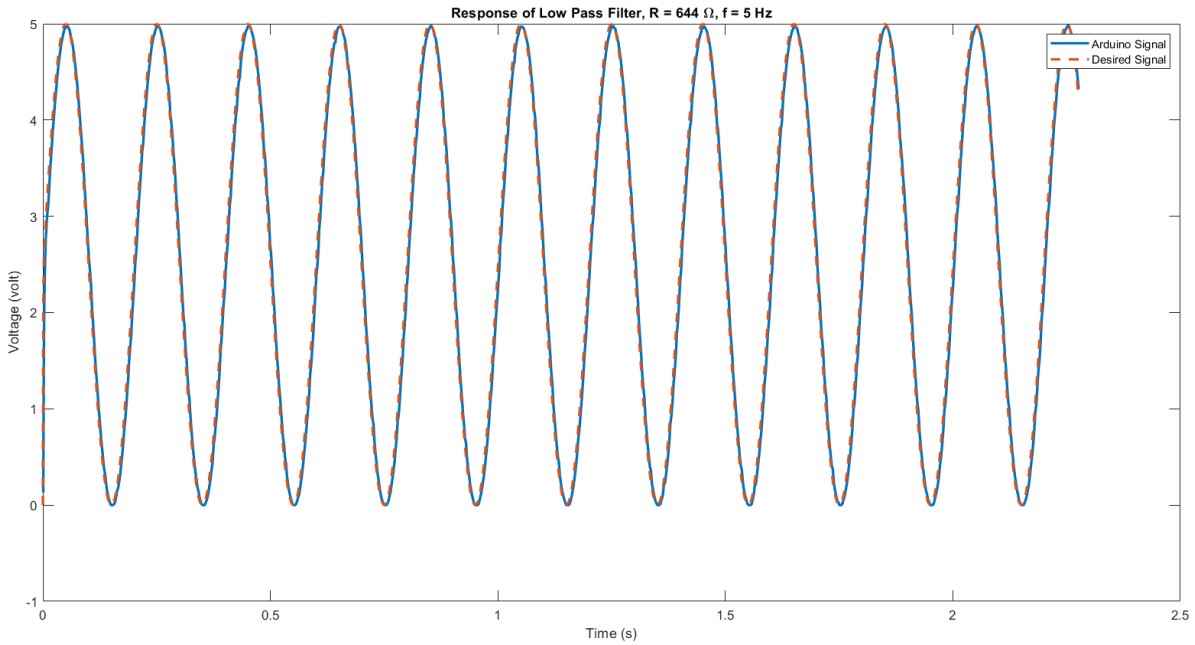

Figure 11: MATLAB result for f = 5 Hz

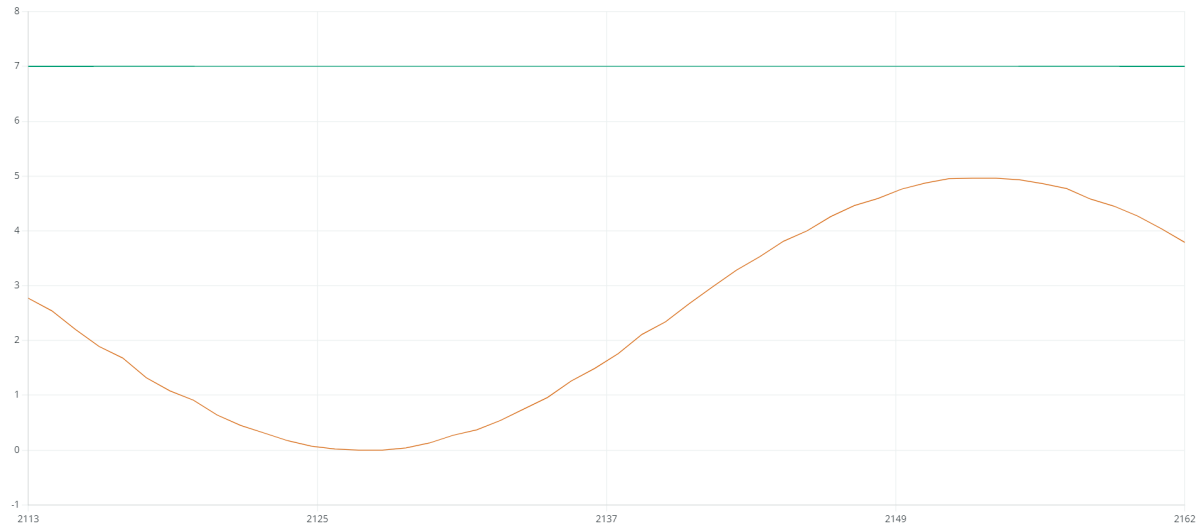

Figure 12: Arduino result for f = 7 Hz

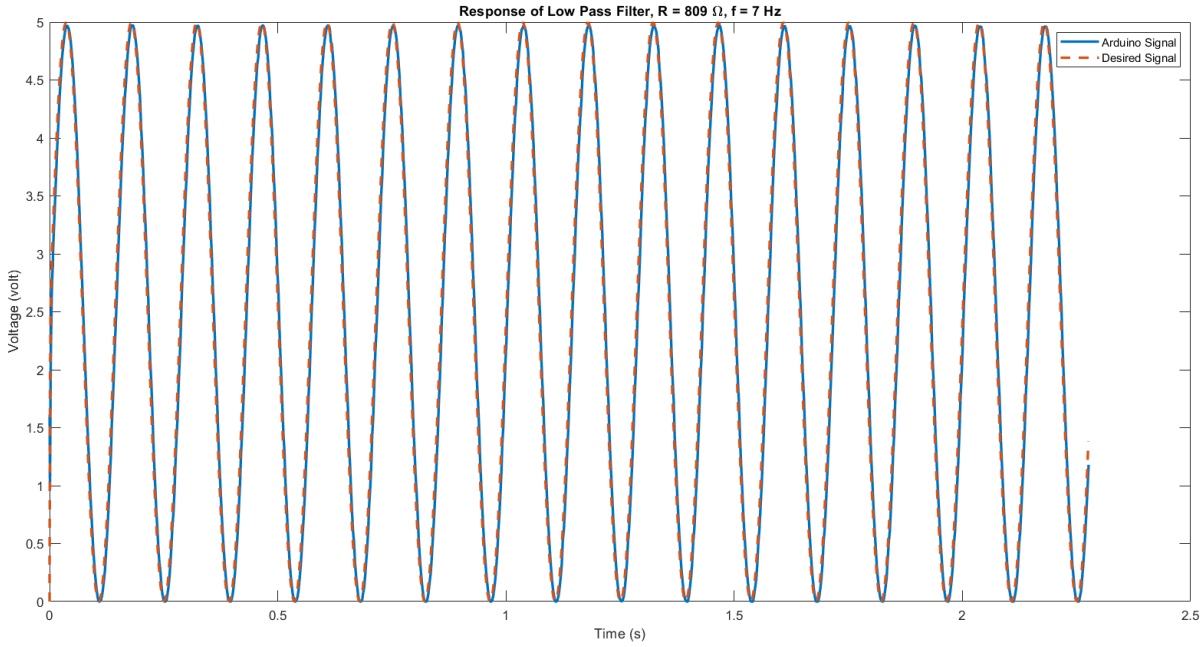

Figure 13: MATLAB result for f = 7 Hz

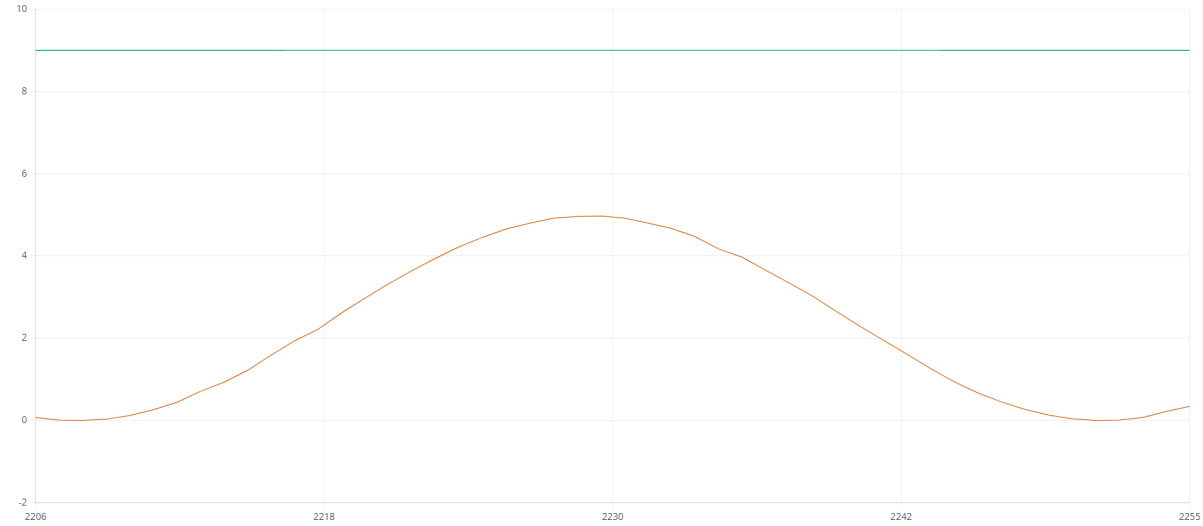

Figure 14: Arduino result for f = 9 Hz

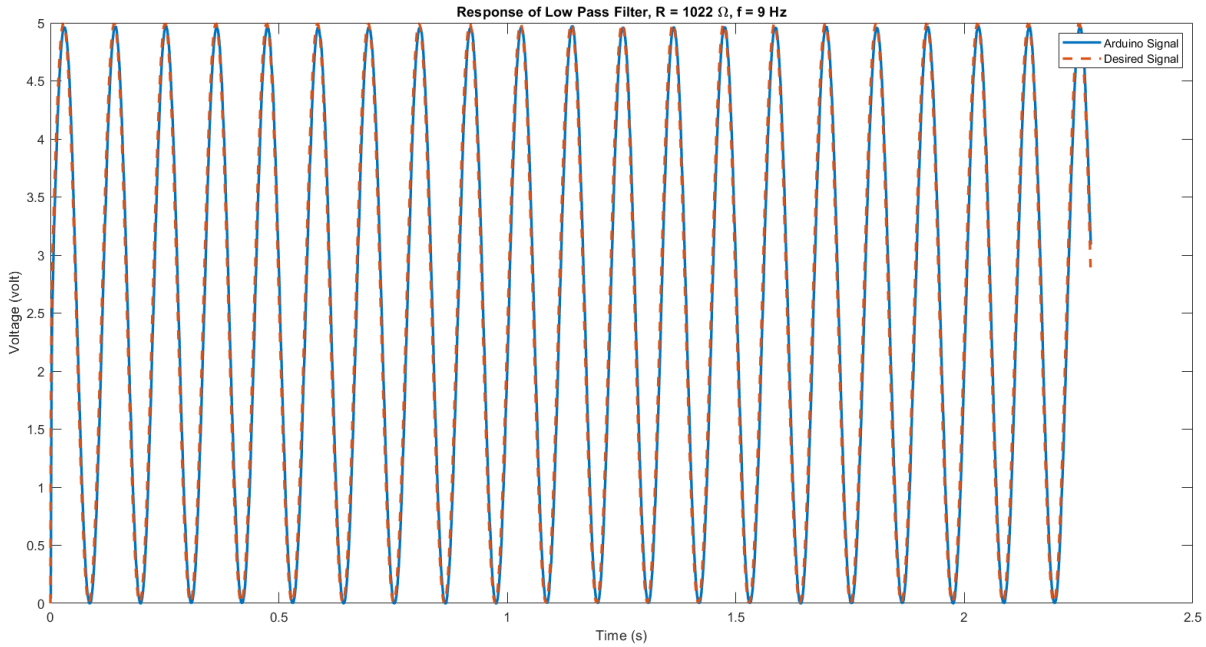

Figure 15: MATLAB result for f = 9 Hz

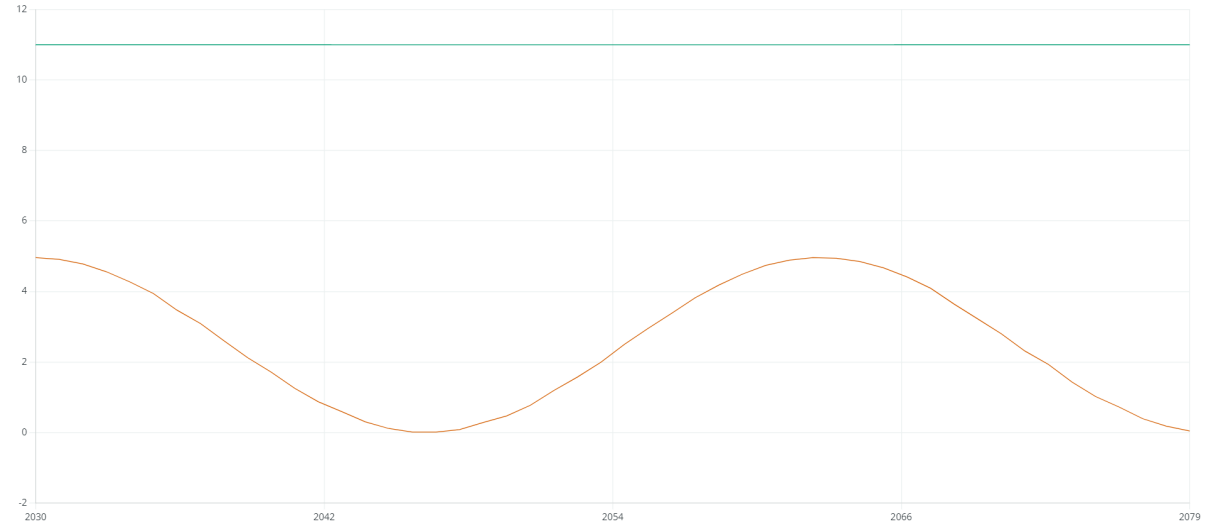

Figure 16: Arduino result for f = 11 Hz

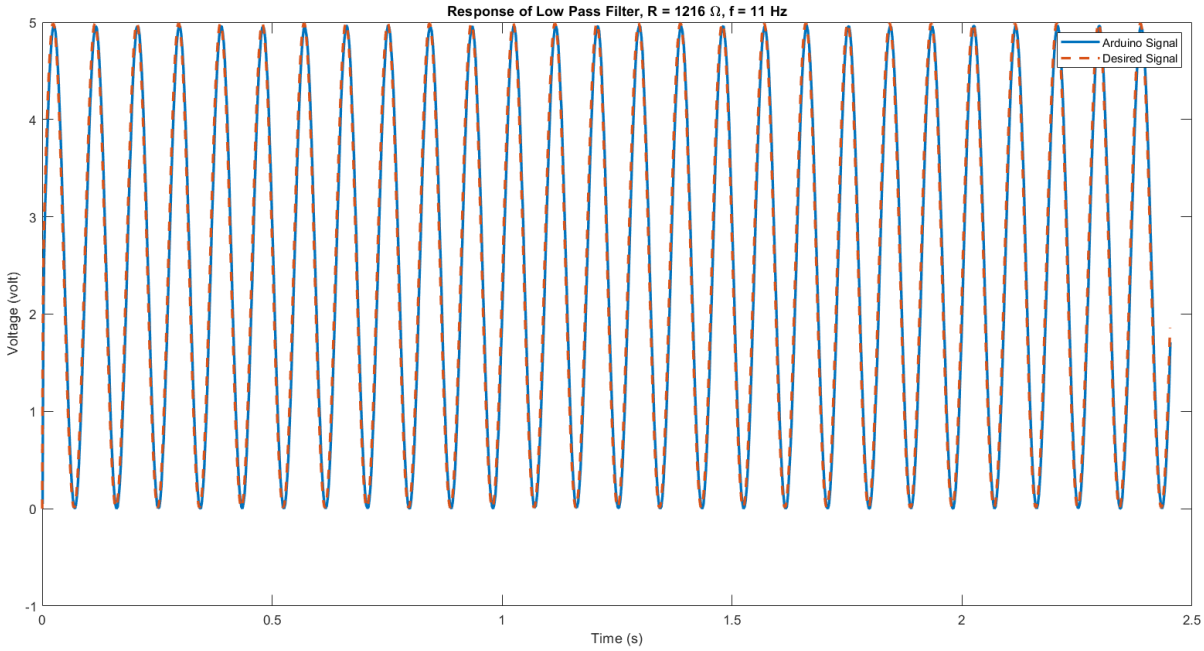

Figure 17: MATLAB result for f = 11 Hz

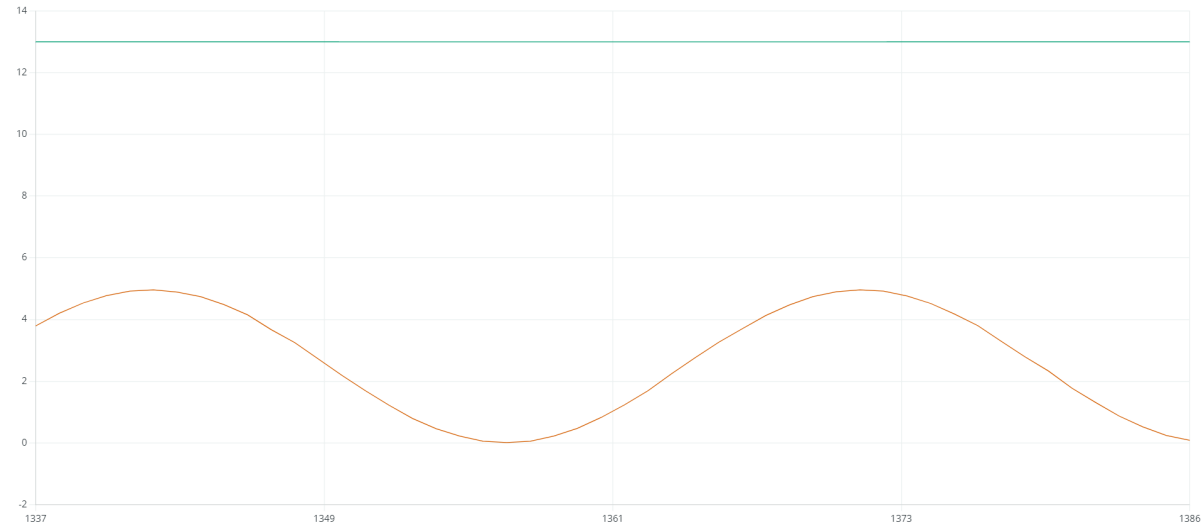

Figure 18: Arduino result for f = 13 Hz

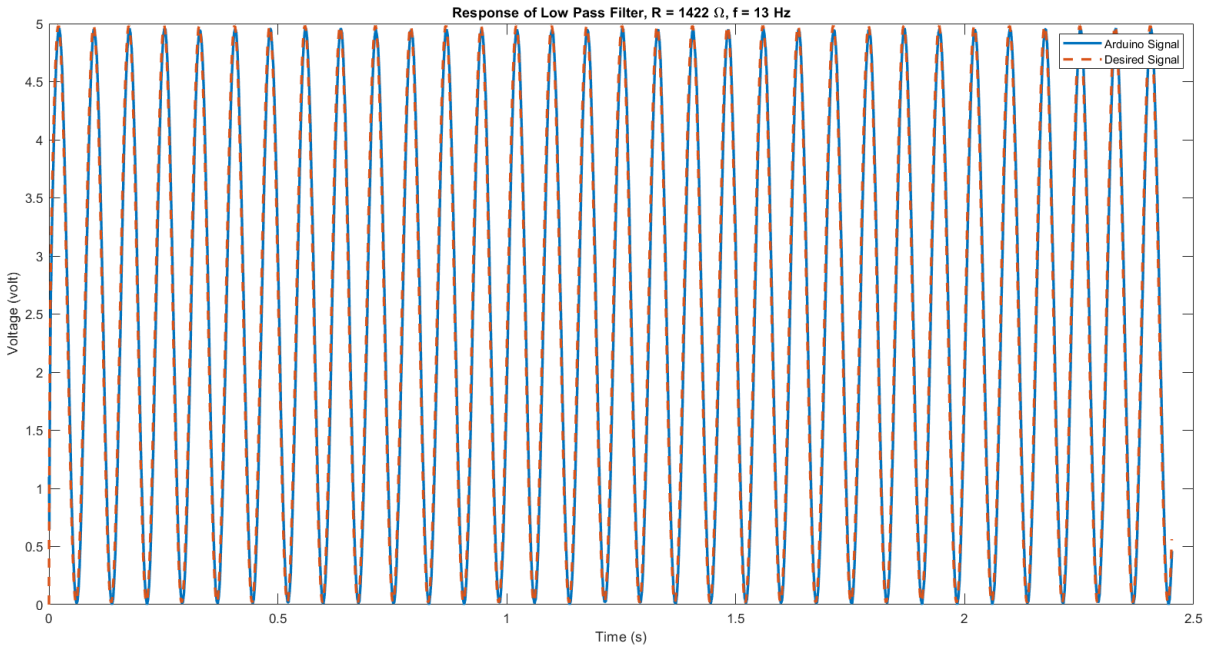

Figure 19: MATLAB result for f = 13 Hz

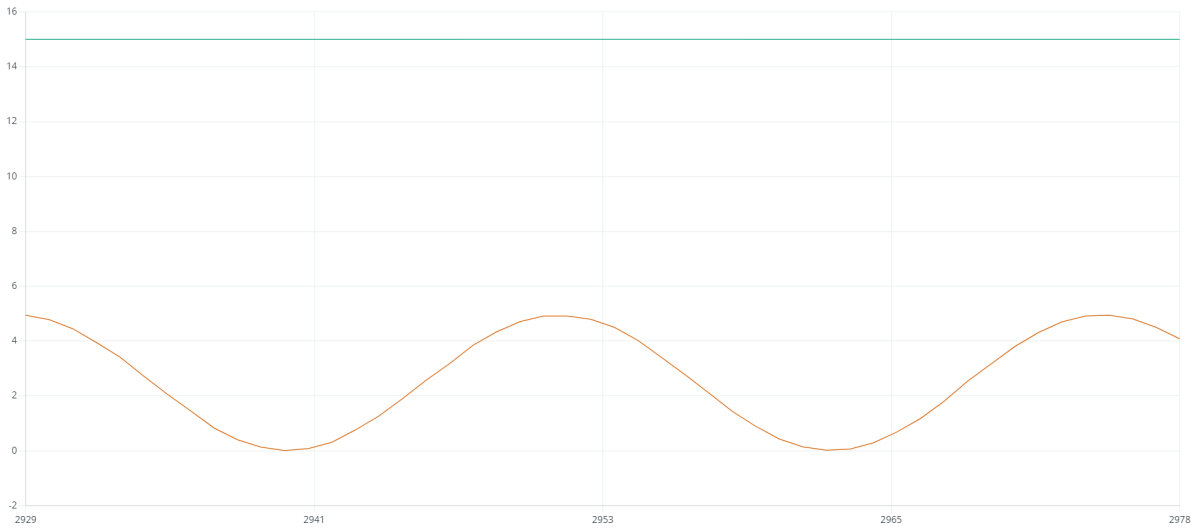

Figure 20: Arduino result for f = 15 Hz

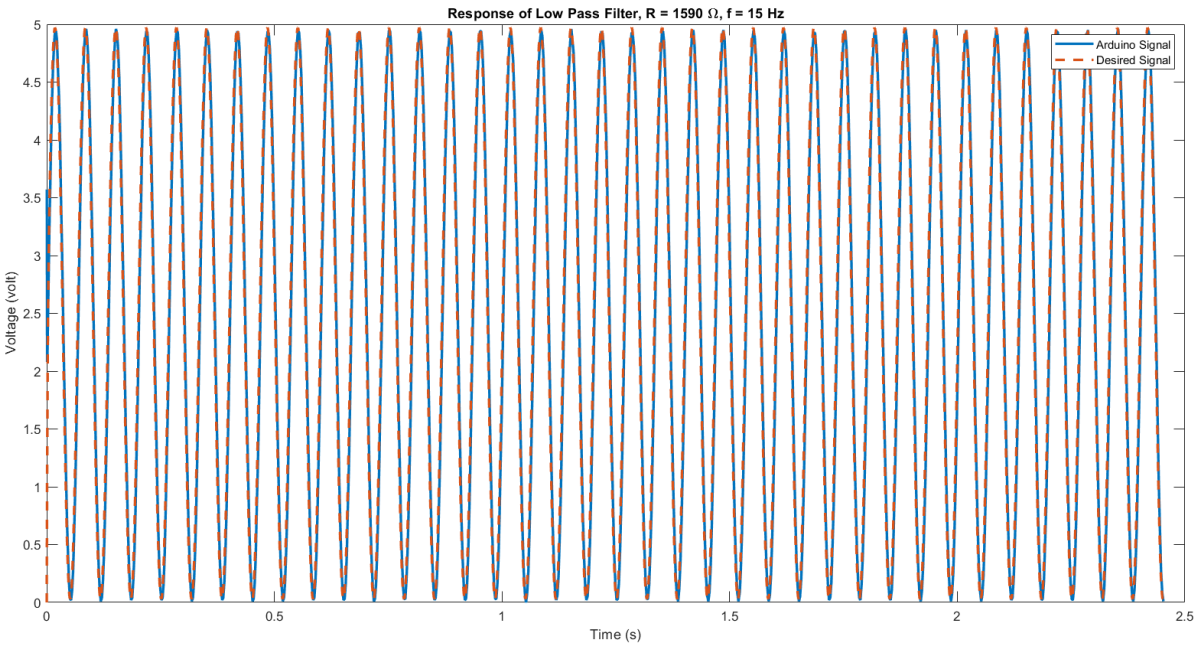

Figure 21: MATLAB result for f = 15 Hz

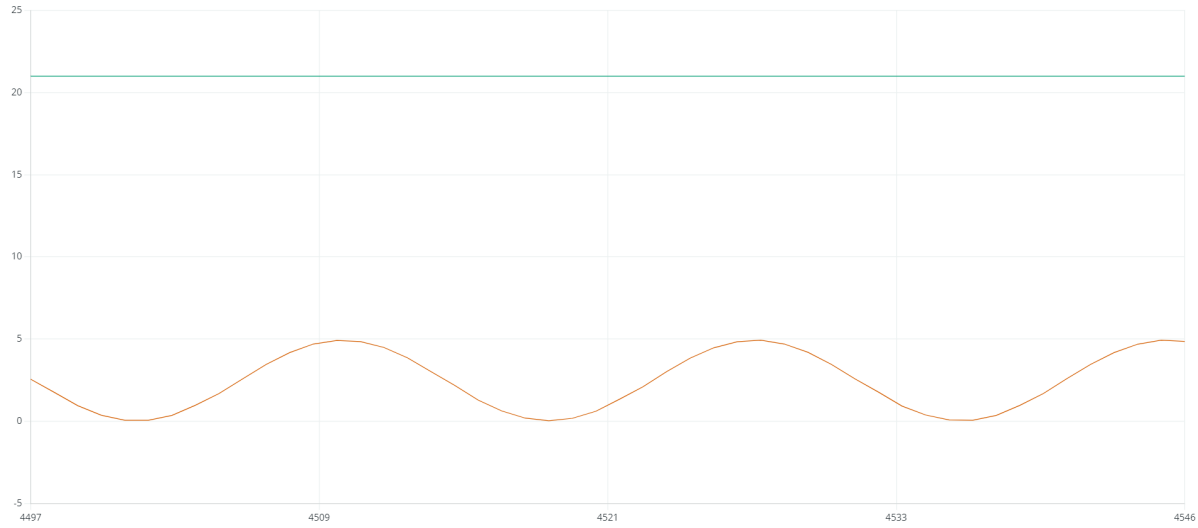

Figure 22: Arduino result for f = 21 Hz

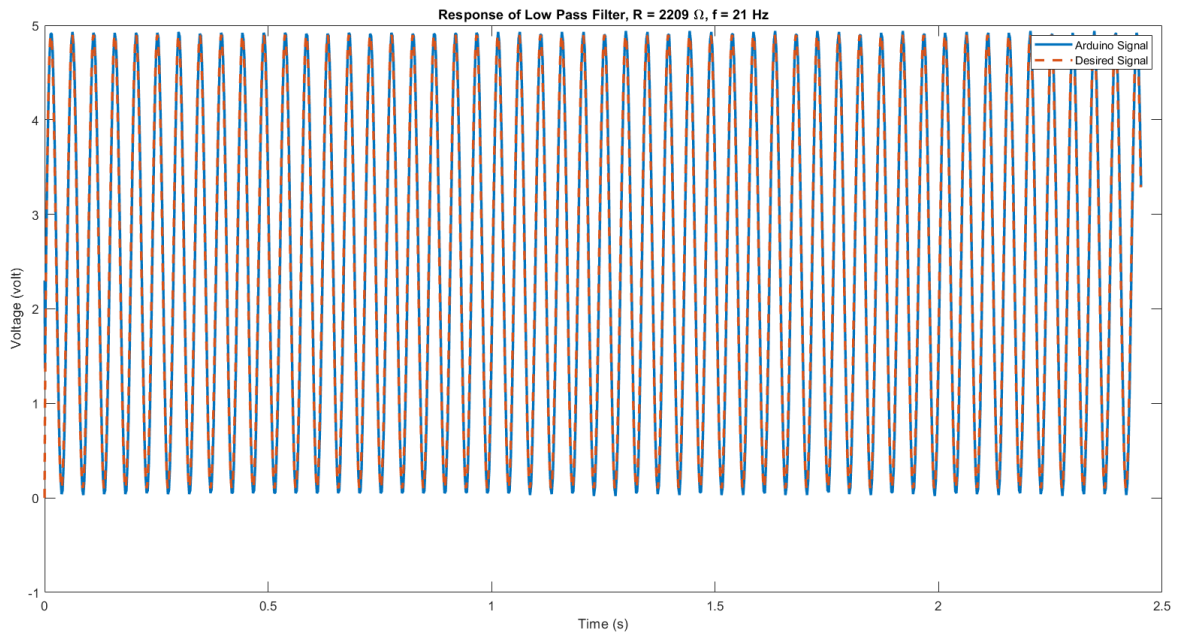

Figure 23: MATLAB result for f = 21 Hz

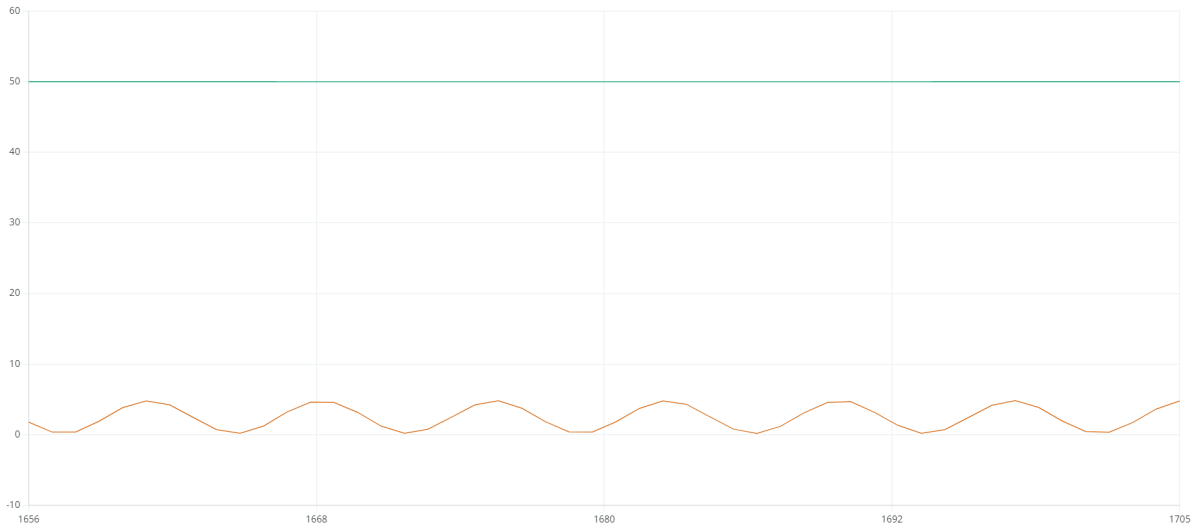

Figure 24: Arduino result for f = 50 Hz

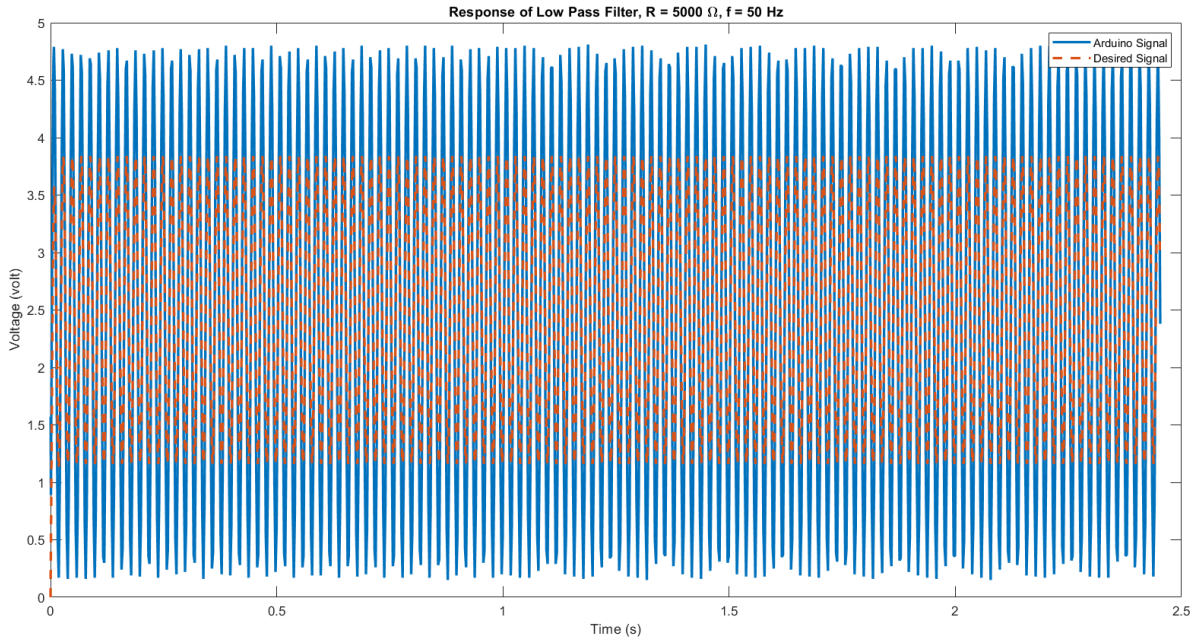

Figure 25: MATLAB result for f = 50 Hz

4 Assignment 2

In this assignment, we were required to build a system capable of generating a signal composed of two sinusoidal components: a low-frequency sine wave and a high-frequency sine wave, both with equal amplitudes. These two signals were added together to form a composite signal in which the higher frequency was meant to be removed using a low-pass filter. The goal was to attenuate the high-frequency component while preserving the low-frequency one, making the filtering effect visible both in the time domain and through calculated attenuation values.

The specific values for the low and high frequencies were determined using a formula based on the student number. After calculating the appropriate frequencies, we designed an RC low-pass filter by choosing suitable values for R and C so that the cut-off frequency would lie between the two signals. The Arduino was programmed to generate the combined signal and send the output to MATLAB using serial communication. MATLAB was used to record and plot the filtered waveform along with the expected signals. Finally, the attenuation of both frequency components was calculated using theoretical formulas and compared to experimental results obtained from the plots.

The low and high frequencies required for this assignment were calculated using the formulas provided:

$$x = \text{Student Number} = 810600050$$

$$\text{Frequency}_{\text{Low}} = \frac{x \bmod 17}{3} + 1 = \frac{810600050 \bmod 17}{3} + 1 = \frac{15}{3} + 1 = 6 \text{ Hz}$$

$$\text{Frequency}_{\text{High}} = \text{Frequency}_{\text{Low}} \times 10^{1.146} = 6 \times 10^{1.146} = 83.9752 \text{ Hz}$$

So the signal consisted of two sine waves: one with a low frequency of approximately **6 Hz** and the other with a high frequency of approximately **83.9752 Hz**, both with equal amplitudes.

In this part, we implemented two different low-pass filters to evaluate their ability to attenuate a high-frequency component from a composite signal. The signal was composed of a 6 Hz low-frequency sine wave and an 83.975 Hz high-frequency sine wave, both generated by the Arduino. The goal was to compare the affects of different filters.

We designed two filters:

- The first filter has a cut-off frequency closer to the **low frequency** component.
- The second filter has a cut-off frequency closer to the **high frequency** component.

Each configuration is explained and analyzed separately below.

4.1 Filter 1: Cut-off Closer to Low Frequency (122 Ω, 100 μF)

In this setup, we used a 100 μF capacitor and a total resistance of 122 Ω (a 22 Ω resistor in series with a 100 Ω resistor). The purpose was to create a filter that strongly suppresses the high-frequency content while preserving the low-frequency sine wave.

The cut-off frequency f_c is calculated as:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 122 \cdot 100 \times 10^{-6}} \approx 13.05 \text{ Hz}$$

This frequency lies well below the high-frequency component (83.975 Hz), so strong attenuation is expected.

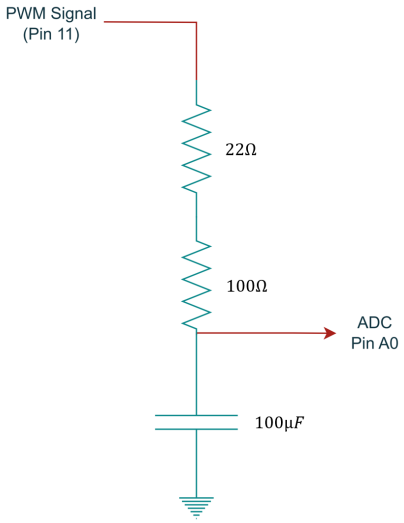

Figure 26: Schematic of Filter 1 – $R = 122\,\Omega$, $C = 100\,\mu$F

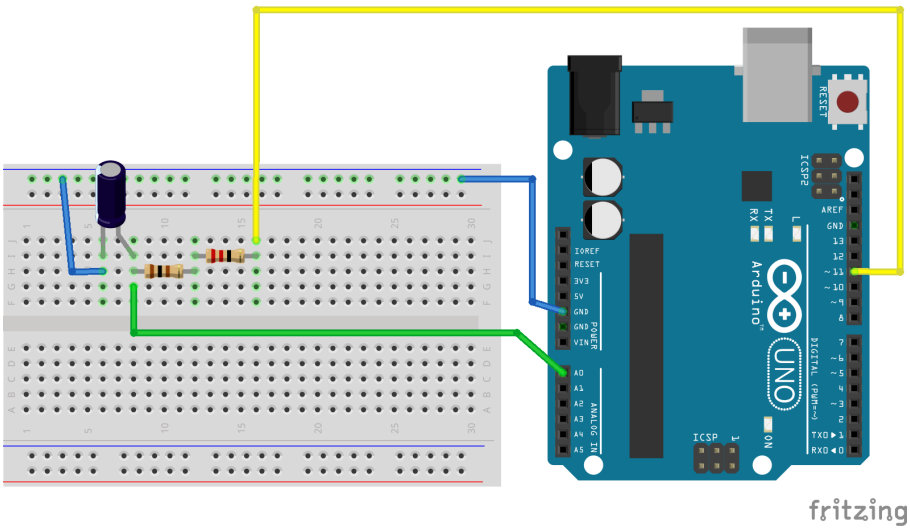

Figure 27: Breadboard implementation of Filter 1 (R = 122 Ω)

As seen in Figures 26 and 27, the circuit was built with the PWM signal from Pin 11 connected through the resistors to a capacitor, which is grounded. The filtered output is measured at the junction between the 100 Ω resistor and the capacitor, and sent to analog pin A0 for data logging and MATLAB analysis.

4.2 Filter 2: Cut-off Closer to High Frequency (33 Ω, 100 μF)

In this configuration, we reduced the resistance to 33 Ω using three 22 Ω resistors (one in series and two in parallel). This raises the cut-off frequency so that the filter is less aggressive and allows more of the high-frequency component through.

The cut-off frequency is calculated as:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 33 \cdot 100 \times 10^{-6}} \approx 48.27 \text{ Hz}$$

This is still below 83.975 Hz, but much closer than the previous filter, so we expect partial attenuation of the high-frequency signal.

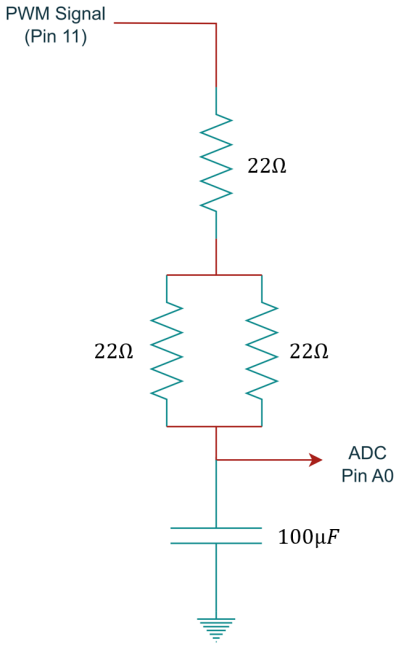

Figure 28: Schematic of Filter 2 – $R = 33\,\Omega$, $C = 100\,\mu$F

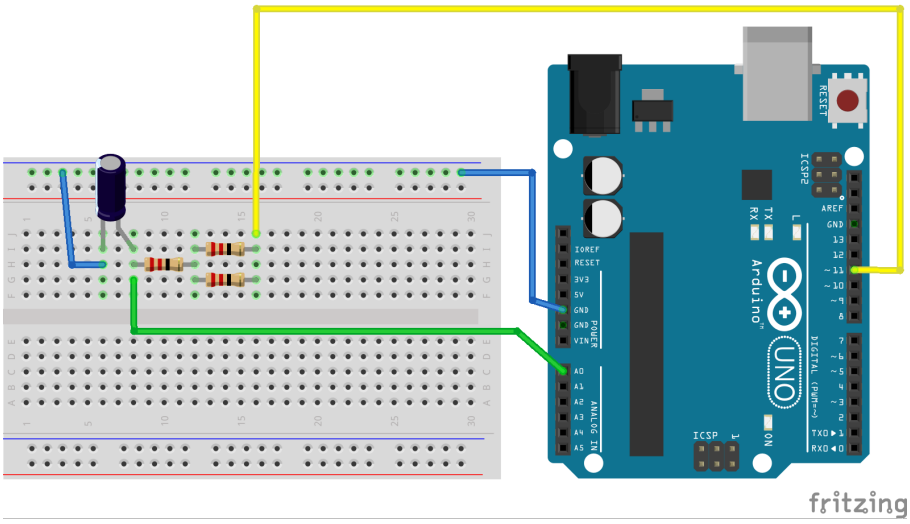

Figure 29: Breadboard implementation of Filter 2 (R = 33 Ω)

Figures 28 and 29 show the updated filter configuration. Again, the filtered output was read by the Arduino on analog pin A0 and sent to MATLAB for visualization and frequency response comparison.

The goal of this experiment is to observe how the choice of cut-off frequency affects the attenuation of the high-frequency component in a mixed signal. The mixed signal contains two sine waves: one low-frequency component at 6 Hz and one high-frequency component at 83.975 Hz. Each filter has a different cut-off frequency, which affects how much of each frequency passes through.

Filter 1 (Cut-off = 13.05 Hz): This filter has a cut-off frequency that is much closer to the low-frequency component. Since 13.05 Hz is only slightly higher than 6 Hz and much lower than 83.975 Hz, we expect the following behavior:

- The 6 Hz signal will pass through mostly unaffected, since it is well below the cut-off frequency.
- The 83.975 Hz component is far beyond the cut-off, so it will be strongly attenuated (filtered out).

This means Filter 1 will produce a signal that closely resembles the original low-frequency sine wave, with very little presence of the high-frequency signal.

Filter 2 (Cut-off = 48.27 Hz): In this case, the cut-off frequency is significantly higher and much closer to the high-frequency component. The expected results are:

- The 6 Hz component will still pass through, but with slightly more attenuation than in Filter 1.
- The 83.975 Hz signal is closer to the cut-off, so only partial attenuation will occur. Some portion of the high-frequency content will remain in the output.

Therefore, Filter 2 is expected to produce a mixed output with both frequencies still present, but with the high-frequency component reduced in amplitude compared to the unfiltered signal.

4.3 MATLAB Resutls

We now present the actual results obtained from the Arduino and MATLAB, showing how the output signals were affected by the two different low-pass filters. In both cases, the plot includes:

- **Blue line:** The filtered signal measured by the Arduino (i.e., actual circuit response).
- **Red line:** The ideal filtered signal simulated in MATLAB using a digital filter for comparison.
- **Yellow line:** The original, unfiltered composite signal consisting of both the low-frequency and high-frequency sine waves.

4.3.1 Filter with Cut-off Frequency of 13 Hz

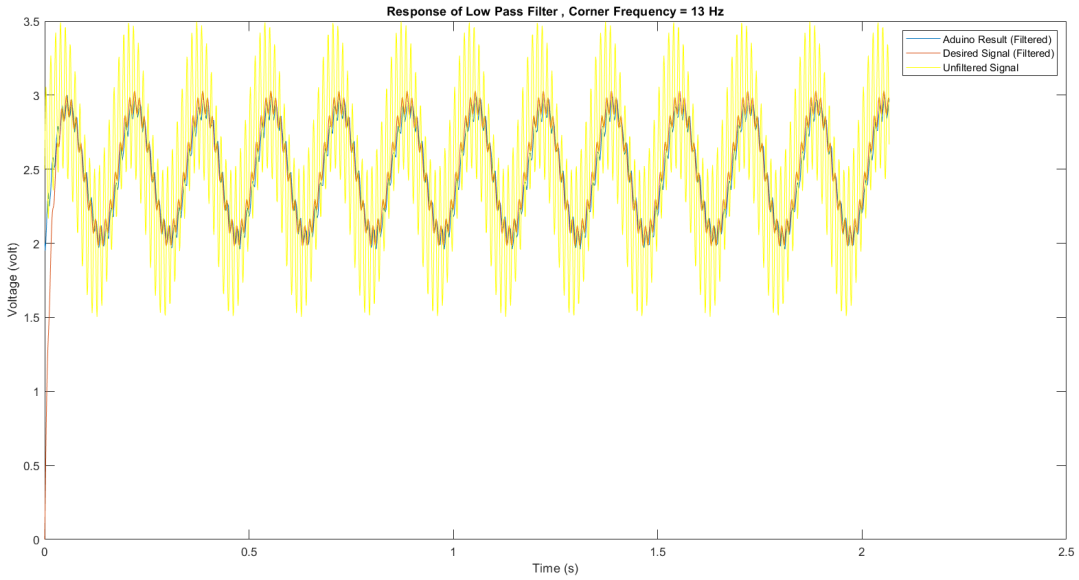

Figure 30: Response of Low-Pass Filter with $f_c = 13\,\text{Hz}$

As seen in Figure 30, the blue curve (Arduino result) closely follows the red curve (ideal MATLAB-filtered signal). Both show a smooth waveform that closely resembles a 6 Hz sine wave, confirming that the filter effectively removed the 83.975 Hz high-frequency component. The yellow curve, representing the original signal, clearly contains rapid high-frequency oscillations that are no longer present in the filtered outputs. This confirms our expectation: since the cut-off frequency of 13 Hz is well below 83.975 Hz, strong attenuation of the high-frequency content was achieved while preserving the low-frequency signal.

4.3.2 Filter with Cut-off Frequency of 48 Hz

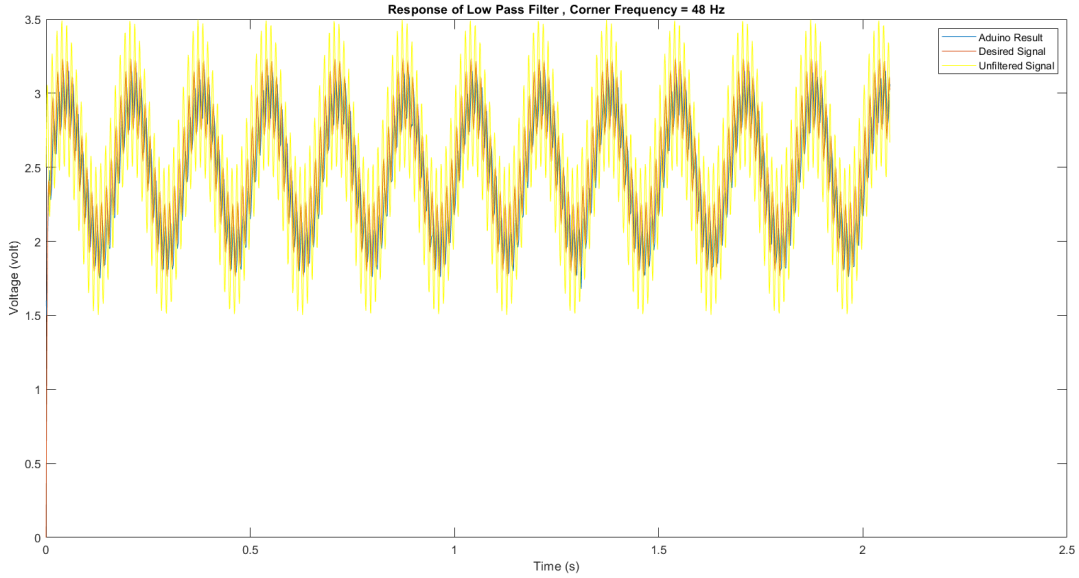

Figure 31: Response of Low-Pass Filter with $f_c = 48\,\text{Hz}$

In Figure 31, the blue and red curves again represent the Arduino result and the ideal MATLAB-filtered signal, respectively. Unlike the previous case, both signals still show noticeable high-frequency content, although less intense than the original yellow waveform. This is because the 48 Hz cut-off frequency is much closer to the high-frequency component (83.975 Hz), so the filter only partially attenuates it. The low-frequency waveform is still visible but is somewhat distorted due to the remaining high-frequency content.

Thus, both filters behave as expected, and the experimental plots confirm the relationship between cut-off frequency and signal attenuation. The results also demonstrate the importance of carefully choosing filter parameters based on the frequency content of the signals to be preserved or removed.

4.4 Theoretical Attenuation Calculations

To quantify the filtering effect of each RC low-pass filter, we calculated the theoretical attenuation at both 6 Hz and 84 Hz using the magnitude response formula of a first-order low-pass filter:

$$|H(f)| = \frac{1}{\sqrt{1 + \left(\frac{f}{f_c}\right)^2}} \qquad \text{and} \qquad \text{Attenuation (dB)} = 20 \cdot \log_{10}(|H(f)|)$$

Filter 1: $f_c = 13\,\textbf{Hz}$

$$|H(6)| = \frac{1}{\sqrt{1 + \left(\frac{6}{13}\right)^2}} \approx 0.906 \Rightarrow \text{Attenuation} = 20 \cdot \log_{10}(0.906) \approx -0.86\,\text{dB}$$

$$|H(84)| = \frac{1}{\sqrt{1 + \left(\frac{84}{13}\right)^2}} \approx 0.151 \Rightarrow \text{Attenuation} = 20 \cdot \log_{10}(0.151) \approx -16.42\,\text{dB}$$

Filter 2: $f_c = 48\,\textbf{Hz}$

$$|H(6)| = \frac{1}{\sqrt{1 + \left(\frac{6}{48}\right)^2}} \approx 0.992 \Rightarrow \text{Attenuation} = 20 \cdot \log_{10}(0.992) \approx -0.07\,\text{dB}$$

$$|H(84)| = \frac{1}{\sqrt{1 + \left(\frac{84}{48}\right)^2}} \approx 0.491 \Rightarrow \text{Attenuation} = 20 \cdot \log_{10}(0.491) \approx -6.18\,\text{dB}$$

Cut-off Frequency	Signal Frequency	Attenuation (dB)
13 Hz	6 Hz	–0.86 dB
13 Hz	84 Hz	–16.42 dB
48 Hz	6 Hz	–0.07 dB
48 Hz	84 Hz	–6.18 dB

Table 1: Theoretical attenuation of each signal frequency for both filter configurations

5 Assignment 3

In this task, we focused on characterizing the frequency response of the RC low-pass filter already used in Assignment 2. To avoid redundancy, no new circuit was built for this part of the experiment the same setup shown in Figures 26 and 27 (with $R = 122\,\Omega$ and $C = 100\,\mu$F) was reused. The goal here was not to design a new filter, but to test and measure its performance over a wide range of frequencies.

The main objective is to experimentally evaluate how the RC low-pass filter attenuates input signals of different frequencies. This is done by:

- Generating sinusoidal signals using Arduino at various frequencies.
- Passing these signals through the existing low-pass filter.
- Measuring the amplitude of the filtered output using the ADC.
- Calculating the attenuation in dB for each input frequency.
- Plotting the frequency response and comparing it with the theoretical prediction.

To achieve this, sinusoidal signals of varying frequencies are generated using Arduino. These signals are then passed through the RC low-pass filter circuit. For each input frequency, the amplitude of the output signal (after the filter) is measured using the analog-to-digital converter (ADC) on the Arduino board.

By comparing the output amplitudes with the known input amplitudes, the attenuation at each frequency is calculated. Attenuation is expressed in decibels (dB) using the following formula:

$$\text{Attenuation (dB)} = 20\log_{10}\left(\frac{V_{\text{out}}}{V_{\text{in}}}\right)$$

This process is repeated for at least 10 different frequencies to create a dataset of attenuation values across the frequency spectrum. Special attention is paid to collecting more data points near the expected cutoff frequency, where the filter's response changes most rapidly.

Finally, the experimental data is used to plot the measured frequency response of the filter. This is compared against the theoretical frequency response derived from the standard transfer function of a first-order RC low-pass filter:

$$|H(f)| = \frac{1}{\sqrt{1 + \left(\frac{f}{f_c}\right)^2}}$$

where $f_c = \frac{1}{2\pi RC}$ is the cutoff frequency of the filter.

This comparison helps students understand how real filters behave, identify deviations due to practical limitations (like non-ideal components or sampling errors), and gain hands-on experience with signal processing and microcontroller-based data acquisition.

5.1 Visual Results and Analysis

The expected output (shown in red dashed line in the plots) was obtained in MATLAB using the system's transfer function:

$$G_{plant}(s) = \frac{1}{RCs + 1}$$

Given:

$$R = 122\ \Omega, \quad C = 100\ \mu\text{F} \quad \Rightarrow \quad \tau = RC = 1.22 \times 10^{-2}\ \text{s}$$

Also, the input voltage was modeled as:

$$V_{\text{in}}(t) = 2.5 \cdot \sin(2\pi f t) + 2.5$$

and we assume that

$$G_{in} = \mathcal{L}\{V_{in}\}$$

After we took the Laplace transform of the input, it is multiplied by G_{plant}, and then used inverse Laplace to compute $V_{\text{out}}(t)$, the desired (theoretical) filtered signal.

$$V_{desired} = \mathcal{L}^{-1}\{G_{plant}.G_{in}\}$$

This response was plotted alongside the actual Arduino measurements to observe how closely the real filter behavior matches the theory.

Each figure below shows the output of the RC filter (blue) and the expected filtered signal (red dashed) for a specific frequency:

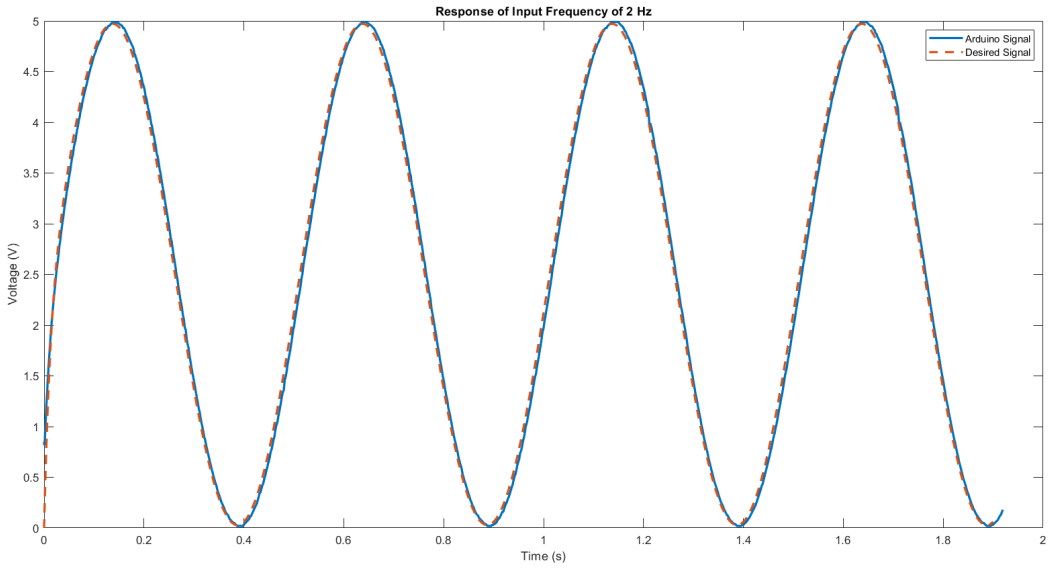

Figure 32: Response of input frequency 2 Hz

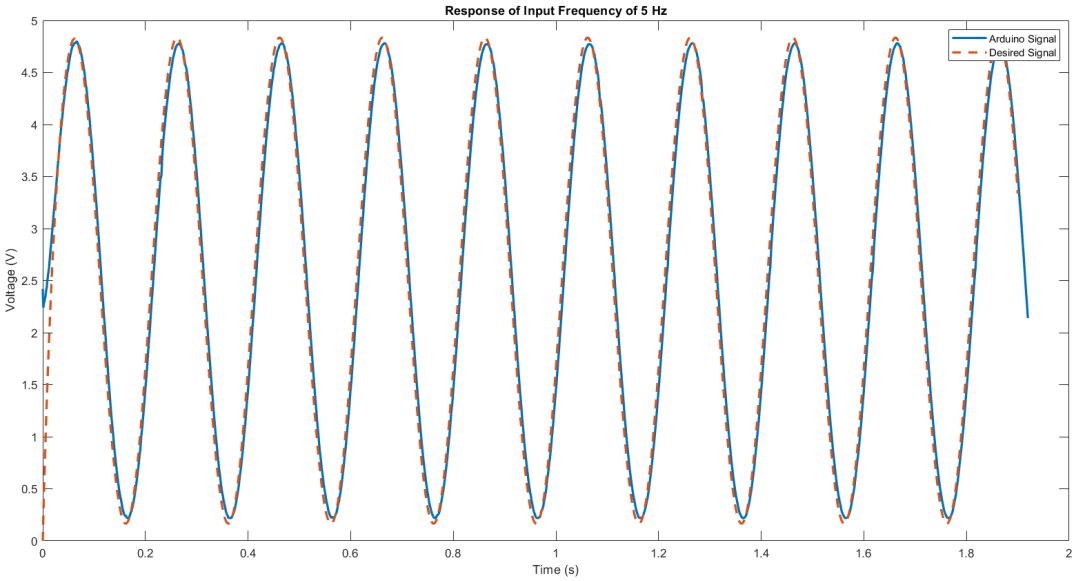

Figure 33: Response of input frequency 5 Hz

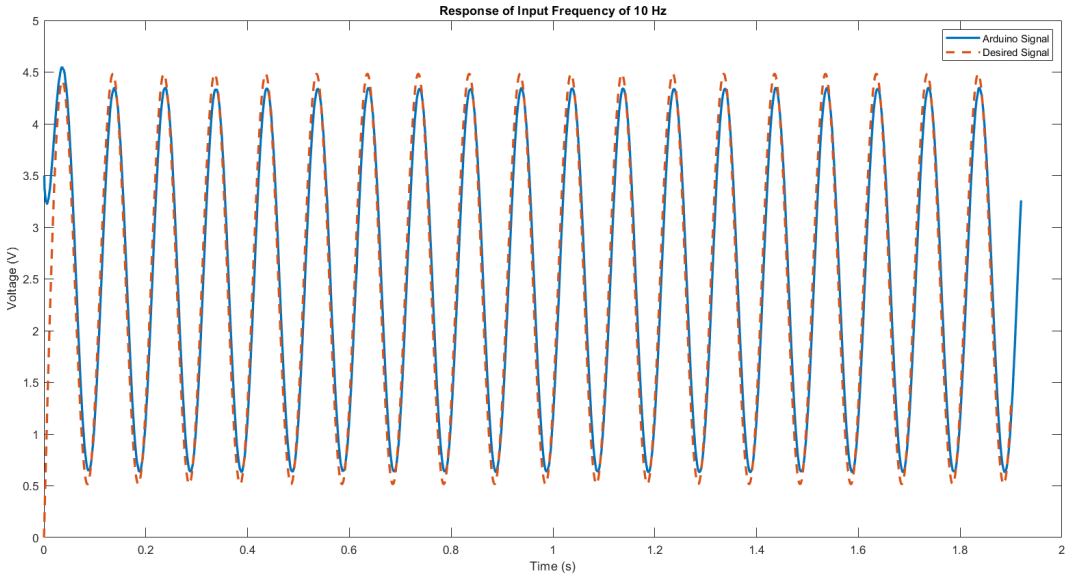

Figure 34: Response of input frequency 10 Hz

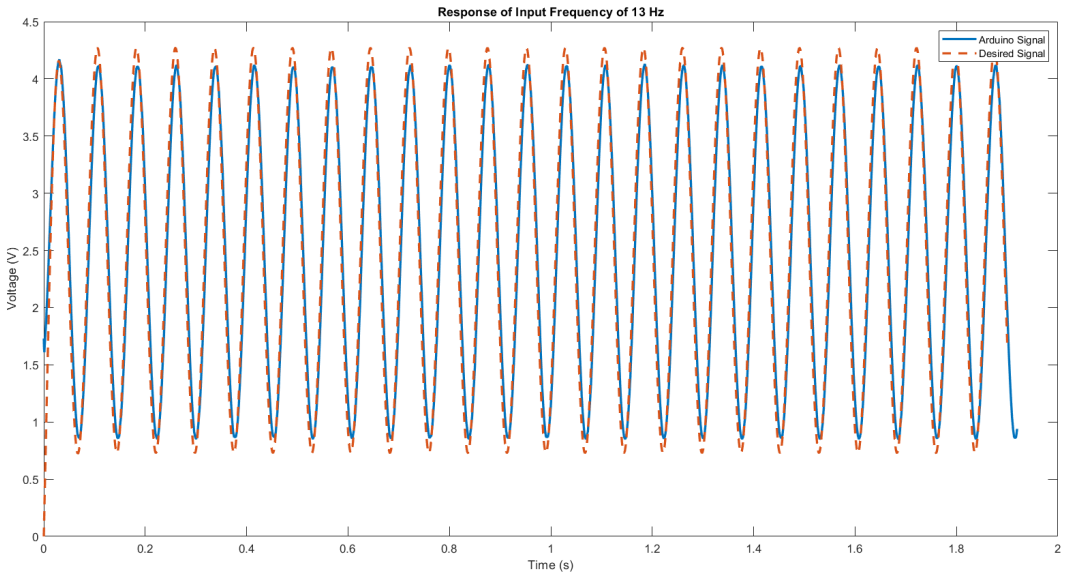

Figure 35: Response of input frequency 13 Hz

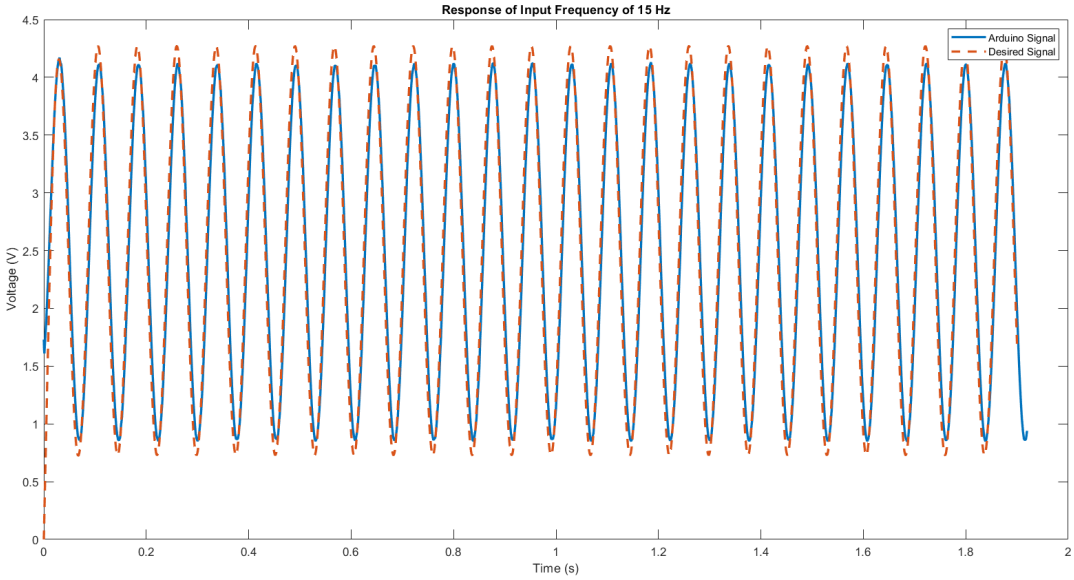

Figure 36: Response of input frequency 15 Hz

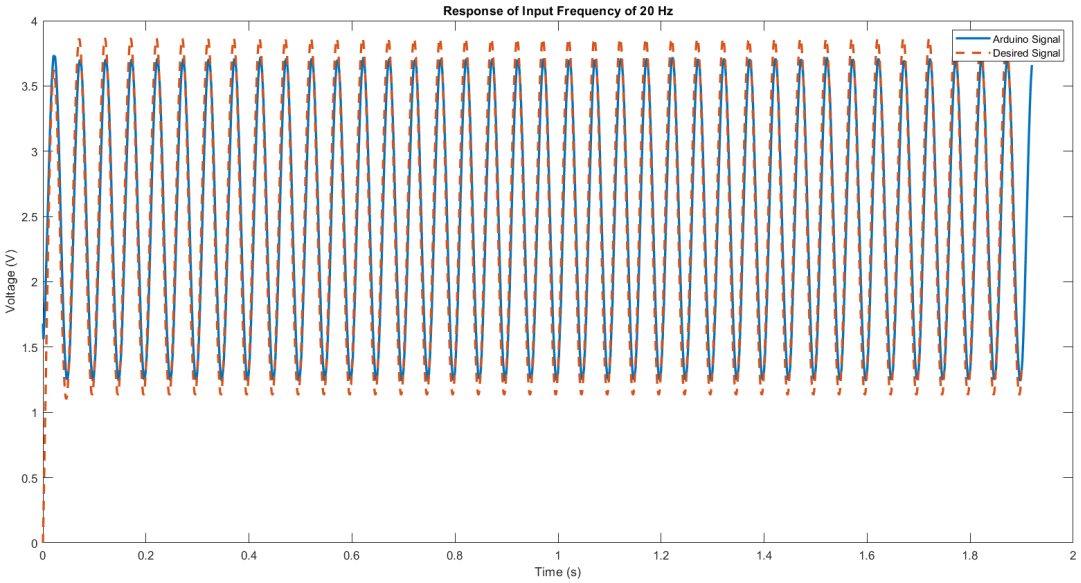

Figure 37: Response of input frequency 20 Hz

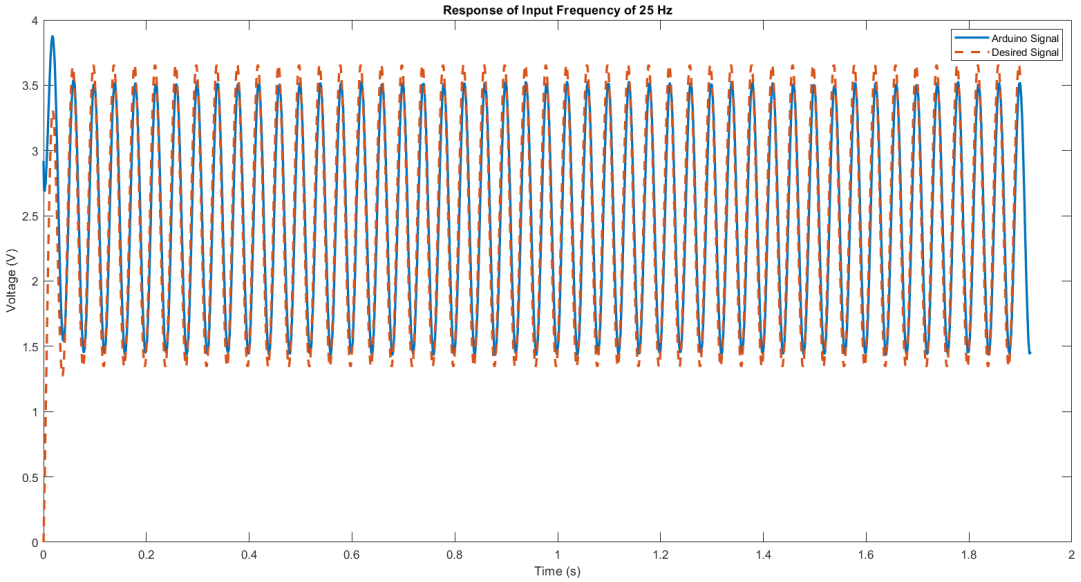

Figure 38: Response of input frequency 25 Hz

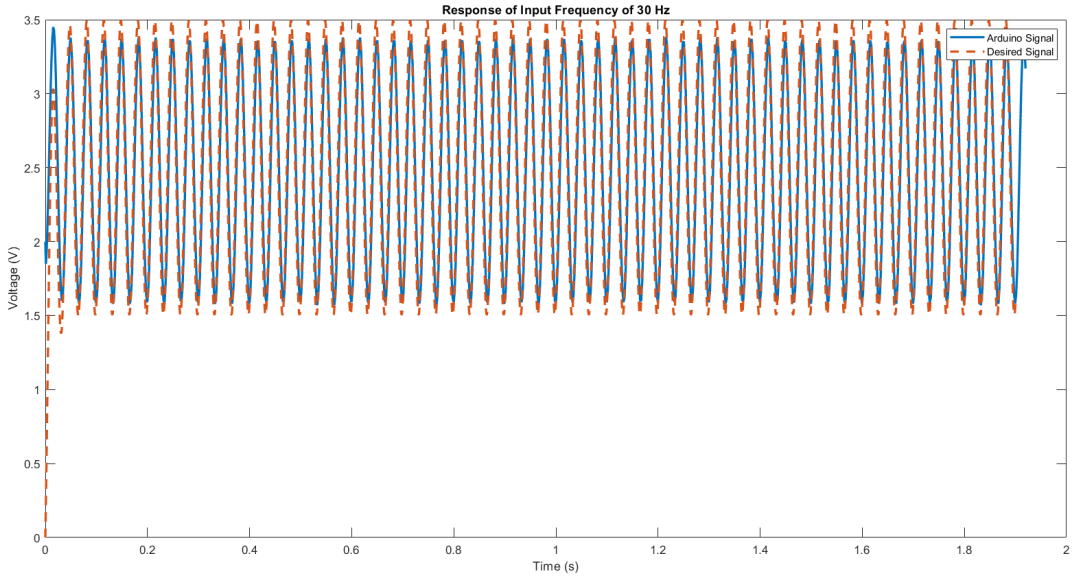

Figure 39: Response of input frequency 30 Hz

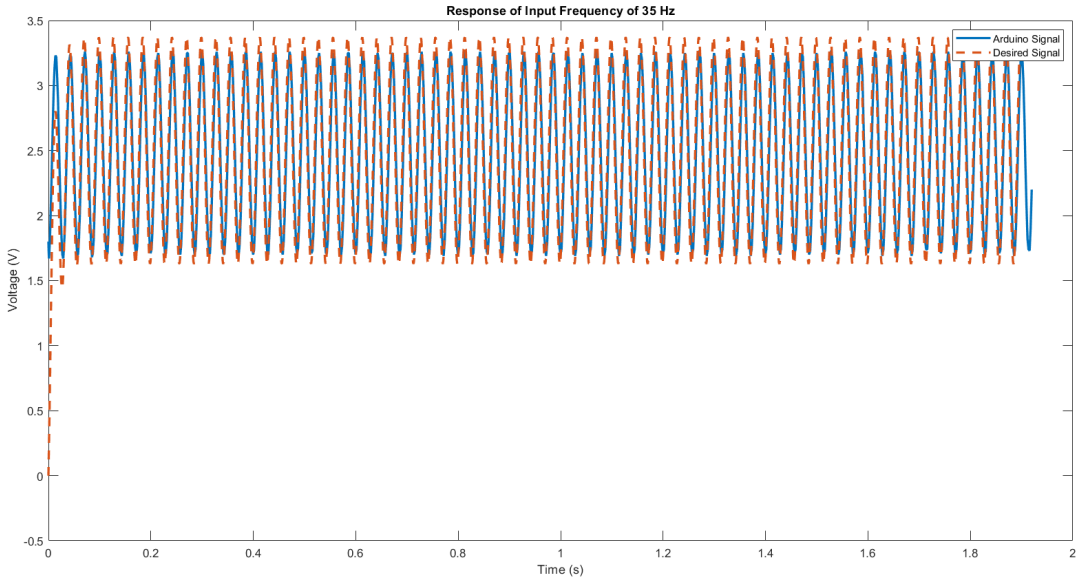

Figure 40: Response of input frequency 35 Hz

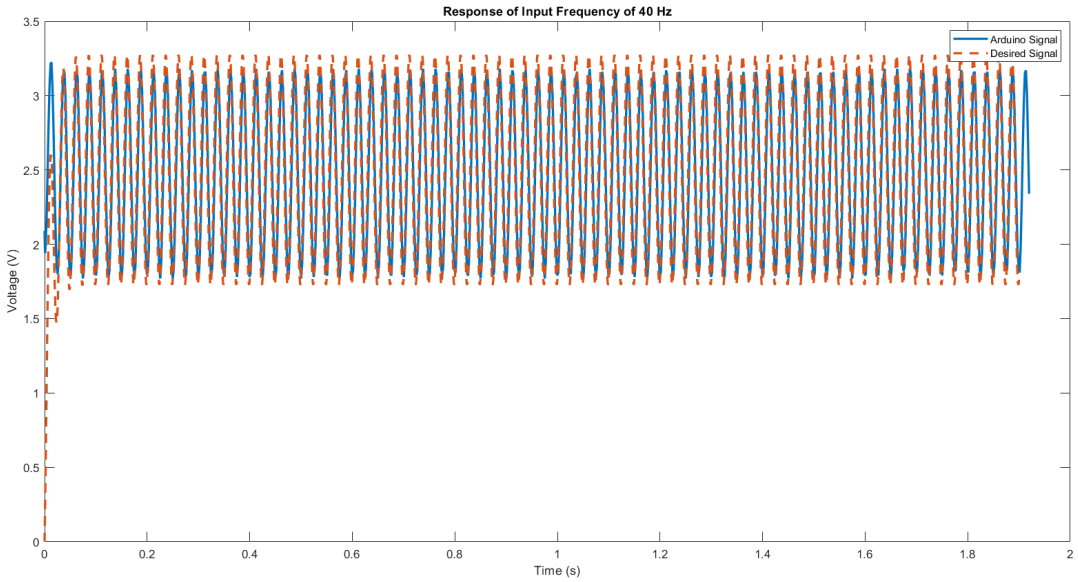

Figure 41: Response of input frequency 40 Hz

5.1.1 Observation Summary

At low frequencies (e.g., 2 Hz, 5 Hz), the Arduino signal closely tracks the theoretical response with almost no distortion, confirming that the filter allows low frequencies to pass. As the frequency increases, attenuation becomes visible. For frequencies near and above the cut-off (13 Hz–30 Hz), the amplitude drops and the output becomes more rounded due to the RC time constant. At 40 Hz, significant attenuation is observed, and the output is heavily smoothed, validating the low-pass behavior.

These results match theoretical expectations and demonstrate the practical performance of a first-order RC filter when built with real components.

5.2 Amplitude Measurement and Attenuation Calculation

To experimentally evaluate the attenuation of the RC low-pass filter, we measured the amplitude of the output signal at various input frequencies. The input sine wave generated by the Arduino had a peak amplitude of approximately 2.5 V. The filtered output was recorded and plotted in MATLAB.

To find the amplitude of the filtered signal at each frequency, we used MATLAB's **Data Cursor (Data Tip)** tool. This tool allowed us to manually inspect the waveform and identify the peak voltage value of the output. The amplitude was taken as the difference between the peak and the mean offset, which is approximately 2.5 V.

Once the amplitude at each frequency was obtained, the **attenuation** in decibels (dB) was calculated using the standard formula:

$$\text{Attenuation (dB)} = 20 \cdot \log_{10}\left(\frac{A_{\text{out}}}{A_{\text{in}}}\right)$$

where:

- A_{out}: measured output amplitude from the filtered signal,
- $A_{\text{in}} = 2.5\,\text{V}$: known input amplitude.

The table below summarizes the frequency response measurements:

Frequency (Hz)	Measured Amplitude (V)	Attenuation (dB)
2	2.50	0.00
5	2.27	-1.11
10	1.86	-2.56
13	1.64	-3.68
15	1.50	-4.44
20	1.23	-6.17
25	1.045	-7.60
30	0.90	-8.74
35	0.78	-9.86
40	0.695	-10.82

Table 2: Attenuation vs. Frequency for RC Filter ($R = 122\,\Omega, C = 100\,\mu\text{F}$)

These results show a clear trend: as the frequency increases, the measured output amplitude decreases and attenuation increases. This confirms the expected low-pass behavior of the filter, which allows low-frequency signals to pass while increasingly attenuating higher-frequency components.

5.3 Bode Diagram Comparison

To evaluate the filter's frequency response over a broader range, a Bode diagram was created comparing both the **experimental attenuation** (obtained from measured amplitudes as shown in table 2) and the **theoretical response** of the RC low-pass filter.

The theoretical transfer function for a first-order RC low-pass filter is:

$$H(f) = \frac{1}{\sqrt{1 + \left(\frac{f}{f_c}\right)^2}}, \quad \text{with } f_c = \frac{1}{2\pi RC}$$

Using this, the magnitude response in decibels is:

$$|H(f)|_{\text{dB}} = 20 \cdot \log_{10} \left(\frac{1}{\sqrt{1 + \left(\frac{f}{f_c}\right)^2}} \right)$$

In MATLAB, this theoretical curve was plotted over a logarithmic frequency range. The experimental results were also plotted using the measured amplitudes from the Arduino outputs at discrete frequencies (2 Hz to 40 Hz). Each amplitude was converted into decibels using:

$$\text{Attenuation (dB)} = 20 \log_{10} \left(\frac{A_{\text{out}}}{A_{\text{in}}} \right)$$

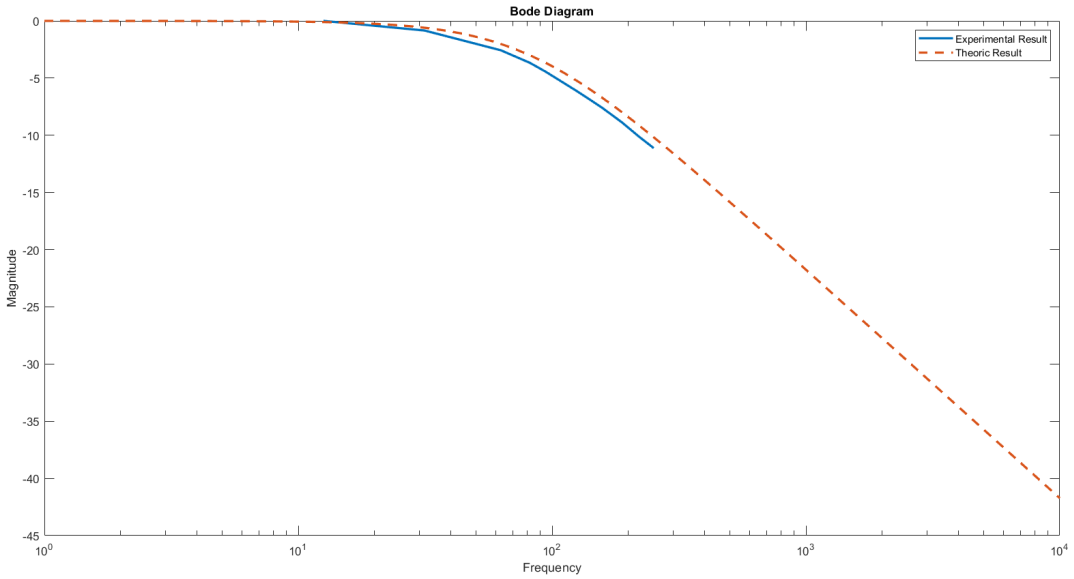

Figure 42: Bode magnitude diagram: Experimental vs. Theoretical response of RC low-pass filter

As shown in Figure 42, the experimental results closely follow the theoretical curve up to approximately the cut-off frequency ($f_c \approx 13\,\text{Hz}$). We can see minor deviations occur in plot, which are mainly due to:

- non-idealities in the components (tolerance and ESR of capacitor),
- limited resolution of the ADC in Arduino (10-bit),
- discretization errors and noise in PWM signal smoothing.

Despite these factors, the measured results validate the expected low-pass behavior. The roll-off slope after the cut-off frequency trends toward the theoretical $-20\,\text{dB/decade}$ characteristic of a first-order filter.

6 Attachments

6.1 Assignment 1 Arduino Code

```
#include <TimerOne.h> // Include the TimerOne library to use Timer1 interrupts

#define Push_Button 2      // Define pushbutton pin
#define PWMOUT 11          // Define PWM output pin
#define ADC_CHANNEL A0     // Define analog input pin for reading filtered
    output

int DUTYCYCLE;             // Variable to store PWM duty cycle
int Sample;                // Variable to store analog read sample
long TimeStamp;            // Variable to store timestamp in microseconds
long Counter = 0;          // Counter used to generate sine wave
int Sine_Freq;             // Current frequency of the sine wave
float Sine_Coeff;          // Coefficient used in sine generation formula
const float Amp = 2.5/5*255;      // Amplitude of sine wave scaled to 8-bit PWM
const float Offset = 2.5/5*255;   // Offset to center sine wave at mid-voltage
const int Max_Freq = 100;         // Maximum allowed frequency

void setup() {
  // Set Timer2 PWM frequency to ~31.37 kHz (faster and less audible)
  TCCR2B = TCCR2B & B11111000 | B00000001;

  // Set PWM output pin as OUTPUT
  pinMode(PWMOUT, OUTPUT);

  // Enable internal pull-up resistor for pushbutton pin
  pinMode(Push_Button, INPUT_PULLUP);

  // Begin serial communication at 57600 baud rate
  Serial.begin(57600);

  // Initialize Timer1 to call an interrupt every 1 millisecond
  Timer1.initialize(1000);

  // Attach the interrupt function PWM_DT to Timer1
  Timer1.attachInterrupt(PWM_DT);

  // Attach an interrupt to the pushbutton pin that triggers on rising edge
  attachInterrupt(digitalPinToInterrupt(Push_Button), Change_Freq, RISING);

  // Read initial frequency value from potentiometer connected to A1
  Sine_Freq = analogRead(A1) / 1023.00 * Max_Freq;

  // Calculate sine wave coefficient based on frequency
  Sine_Coeff = Sine_Freq * TWO_PI / 1000.00;
}

void loop() {
  // Get current time in microseconds
  TimeStamp = micros();

  // Read filtered analog output from the RC circuit
  Sample = analogRead(ADC_CHANNEL);

  // Send time, voltage (in volts), and frequency to serial monitor for MATLAB
  Serial.print(TimeStamp / 100);          // Divide to reduce data size
  Serial.print("\t");
  Serial.print(Sample / 1023.00 * 5);     // Convert ADC value to voltage
```

```
  Serial.print("\t");
  Serial.println(Sine_Freq);              // Print current frequency
}

// Interrupt function that updates the frequency when button is pressed
void Change_Freq() {
  // Read potentiometer value and convert to frequency
  Sine_Freq = analogRead(A1) / 1023.00 * Max_Freq;

  // Update sine coefficient for the new frequency
  Sine_Coeff = Sine_Freq * TWO_PI / 1000.00;
}

// Timer1 interrupt function to update PWM output
void PWM_DT() {
  // Generate sine wave and scale it to 8-bit PWM range (0  255  )
  DUTYCYCLE = Amp * sin(Sine_Coeff * Counter) + Offset;

  // Output the calculated PWM value
  analogWrite(PWMOUT, DUTYCYCLE);

  // Move to next sample point
  Counter++;
}
```

6.2 Assignment 1 MATLAB Code

The following code is according to the last data which is Figure 25

```
    clc, clear
N = 1e3;
a = strings(N,3);

s = serialport("COM7",57600);
configureTerminator(s,"CR/LF");
for i = 1:N
    a(i,:) = strsplit(readline(s));
end
b = double(a);
clear s


syms s t
plot((b(:,1))/1e4,b(:,2),'LineWidth',2)
hold on
f = 50;
y = 2.5*sin(2*pi*f*t)+2.5;
G_s = laplace(y);
R = 5000;
C = 1e-6;
G_filter = 1/(1 + R*C*s);
f = vpa(ilaplace(G_filter*G_s,t));
tt = 0 : 0.001 : 2.4527;
plot(tt,subs(f,t,tt),'--','LineWidth',2)
xlabel('Time (s)')
ylabel('Voltage (volt)')
title('Response of Low Pass Filter, R = 5000 \Omega, f = 50 Hz');
legend('Arduino Signal','Desired Signal')
```

6.3 Assignment 2 Arduino Code

```
    /* Assignment 2
 * Amirhossein Ansari
 * Hana Shamsaei
*/

#include <TimerOne.h>  // Include TimerOne library to use hardware Timer1 for
    precise timing

#define PWMOUT 11           // Define pin 11 as the PWM output pin
#define ADC_CHANNEL A0      // Define analog pin A0 as the input to read the
    filtered signal

int DUTYCYCLE;              // Variable to store the current PWM duty cycle
int Sample;                 // Variable to store the analog value read from the
    filter output
long TimeStamp;             // Variable to store the time in microseconds (
    divided for efficiency)
long Counter = 0;           // Counter used to calculate time progression in
    sine wave generation

// Define frequencies of low and high sine wave components
int Sine_Freq_Low = 6;      // Low-frequency sine wave (Hz)
int Sine_Freq_High = 84;    // High-frequency sine wave (Hz)

// Convert frequencies to angular velocity in rad/ms (TWO_PI = 2  )
float Sine_Coeff_Low = Sine_Freq_Low * TWO_PI / 1000.00;
float Sine_Coeff_High = Sine_Freq_High * TWO_PI / 1000.00;

// Scale amplitude (0.5V over 5V full scale, converted to 8-bit PWM range)
const float Amp = 0.5 / 5.00 * 255;

// Offset to center the signal around 2.5V in 8-bit PWM range
const float Offset = 2.5 / 5.00 * 255;

const int Max_Freq = 100;  // Maximum frequency placeholder (not used in this
    code)

void setup() {
  // Change Timer2 prescaler to set PWM frequency to ~31.37 kHz
  TCCR2B = TCCR2B & B11111000 | B00000001;

  pinMode(PWMOUT, OUTPUT);        // Set the PWM pin as output

  Serial.begin(57600);            // Initialize serial communication at 57600
      baud rate

  Timer1.initialize(1000);        // Configure Timer1 to trigger every 1
      millisecond (1000 microseconds)

  Timer1.attachInterrupt(PWM_DT); // Attach PWM_DT function to Timer1 interrupt
}

void loop() {
  TimeStamp = micros() / 100;          // Read current time and scale down for
      compact output
  Sample = analogRead(ADC_CHANNEL);    // Read the filtered analog signal from
      the RC filter

  // Send timestamp and voltage value (scaled to 0  5V  range) to serial output
  Serial.print(TimeStamp);
  Serial.print("\t");
```

```
  Serial.println(Sample / 1023.00 * 5);
}

void PWM_DT() {
  // Generate two sine waves (low and high frequency), sum them, apply scaling
      and offset
  DUTYCYCLE = Amp * sin(Sine_Coeff_Low * Counter) +
              Amp * sin(Sine_Coeff_High * Counter) +
              Offset;

  analogWrite(PWMOUT, DUTYCYCLE); // Output the composite signal as PWM

  Counter++; // Increment counter to progress time in sine calculations
}
```

6.4 Assignment 2 MATLAB Code

```
    load('Ass2_f13.mat')
x = double(Ass2_f13(2:1000,1)).*1e-4;
y = double(Ass2_f13(2:1000,2));
clear t
syms s t
plot(x,y)
hold on
xlabel('Time (s)')
ylabel('Voltage (volt)')
title('Response of Low Pass Filter , Corner Frequency = 13 Hz');
tt = 0:0.001:2.0663;
desired = 0.5*sin(2*pi*6*t)+0.5*sin(2*pi*84*t)+2.5;
G1 = laplace(desired);
G = 1/(122*1e-4*s+1);
f(t) = simplify(ilaplace(G1*G,t));
plot(tt,f(tt));
y1 = 0.5*sin(2*pi*6*tt);
y2 = 0.5*sin(2*pi*84*tt);
final = y1+y2+2.5;
plot(tt,final,'Color','yellow','LineWidth',0.25)
legend('Aduino Result (Filtered)','Desired Signal (Filtered)','Unfiltered Signal')
hold off

load('Ass2_f48.mat')
x = double(Ass2_f48(2:1000,1)).*1e-4;
y = double(Ass2_f48(2:1000,2));
clear t
syms s t
plot(x,y)
hold on
xlabel('Time (s)')
ylabel('Voltage (volt)')
title('Response of Low Pass Filter , Corner Frequency = 48 Hz');
tt = 0:0.001:2.0663;
desired = 0.5*sin(2*pi*6*t)+0.5*sin(2*pi*84*t)+2.5;
G1 = laplace(desired);
G = 1/(33*1e-4*s+1);
f(t) = simplify(ilaplace(G1*G,t));
plot(tt,f(tt));
y1 = 0.5*sin(2*pi*6*tt);
y2 = 0.5*sin(2*pi*84*tt);
final = y1+y2+2.5;
plot(tt,final,'Color','yellow','LineWidth',0.25)
legend('Aduino Result','Desired Signal','Unfiltered Signal')
hold off
```

6.5 Assignment 3 Arduino Code

```
    /* Assignemnt 2
 * Amirhossein Ansari
 * Hana Shamsaei
*/

#include <TimerOne.h>

#define PWMOUT 11
#define ADC_CHANNEL A0

int DUTYCYCLE;
int Sample;
long TimeStamp;
long Counter = 0;
int Sine_Freq = 15;
float Sine_Coeff = Sine_Freq*TWO_PI/1000.00;
const float Amp = 2.5/5.00*255;
const float Offset = 2.5/5.00*255;
const int Max_Freq = 100;


void setup() {
  TCCR2B = TCCR2B & B11111000 | B00000001;  // Sets Timer2 PWM frequency to 31
      KHz
  pinMode(PWMOUT, OUTPUT);
  Serial.begin(57600);
  Timer1.initialize(1000); // 1000 us == 1 ms
  Timer1.attachInterrupt(PWM_DT);
}

void loop() {
  TimeStamp = micros()/100;
  Sample = analogRead(ADC_CHANNEL);

  Serial.print(TimeStamp);    Serial.print("\t");     Serial.println(Sample
      /1023.00*5);
}

void PWM_DT(){
  DUTYCYCLE = Amp*sin(Sine_Coeff*Counter) + Offset;
  analogWrite(PWMOUT, DUTYCYCLE);
  Counter++;
}
```

6.6 Assignment 3 MATLAB Code

6.6.1 Saving Data From Arduino

```
clc, clear
N = 1e3;
a = strings(N,2);

s = serialport("COM8",57600);
configureTerminator(s,"CR/LF");
for i = 1:N
    a(i,:) = strsplit(readline(s));
end
Ass3_f40 = double(a);
clear s

plot(Ass3_f40(:,1)/10000,Ass3_f40(:,2))
```

6.6.2 Plot Data and Bode

```
    clc, clear, close all
load('Assignment3_Data.mat')
syms s t
f = [2; 5; 10; 13; 15; 20; 25; 30; 35; 40];
Mag = [2.50; 2.27; 1.86; 1.64; 1.5; 1.23;
    1.045; 0.9; 0.78; 0.695]/2.5;

G_plant = 1/(122*1e-4*s+1);

% frequency = 2 Hz
plot(Ass3_f2(:,1)/1e4, Ass3_f2(:,2), "LineWidth",2);
hold on

desired_1 = 2.5*sin(2*pi*f(1)*t) + 2.5;
G_input_1 = laplace(desired_1);
y_f1(t) = ilaplace(G_input_1*G_plant);
tt = 0:0.001:1.9;
plot(tt, y_f1(tt), "--", "LineWidth", 2);
title("Response of Input Frequency of 2 Hz");
xlabel("Time (s)");
ylabel("Voltage (V)");
legend("Arduino Signal", "Desired Signal")
hold off
plot(Ass3_f5(:,1)/1e4, Ass3_f5(:,2), "LineWidth",2);
hold on

desired_2 = 2.5*sin(2*pi*f(2)*t) + 2.5;
G_input_2 = laplace(desired_2);
y_f2(t) = ilaplace(G_input_2*G_plant);
tt = 0:0.001:1.9;
plot(tt, y_f2(tt), "--", "LineWidth", 2);
title("Response of Input Frequency of 5 Hz");
xlabel("Time (s)");
ylabel("Voltage (V)");
legend("Arduino Signal", "Desired Signal")
```

```
hold off
plot(Ass3_f10(:,1)/1e4, Ass3_f10(:,2), "LineWidth",2);
hold on

desired_3 = 2.5*sin(2*pi*f(3)*t) + 2.5;
G_input_3 = laplace(desired_3);
y_f3(t) = ilaplace(G_input_3*G_plant);
tt = 0:0.001:1.9;
plot(tt, y_f3(tt), "--", "LineWidth", 2);
title("Response of Input Frequency of 10 Hz");
xlabel("Time (s)");
ylabel("Voltage (V)");
legend("Arduino Signal", "Desired Signal")
hold off


plot(Ass3_f13(:,1)/1e4, Ass3_f13(:,2), "LineWidth",2);
hold on

desired_4 = 2.5*sin(2*pi*f(4)*t) + 2.5;
G_input_4 = laplace(desired_4);
y_f4(t) = ilaplace(G_input_4*G_plant);
tt = 0:0.001:1.9;
plot(tt, y_f4(tt), "--", "LineWidth", 2);
title("Response of Input Frequency of 13 Hz");
xlabel("Time (s)");
ylabel("Voltage (V)");
legend("Arduino Signal", "Desired Signal")
hold off
plot(Ass3_f15(:,1)/1e4, Ass3_f15(:,2), "LineWidth",2);
hold on

desired_4 = 2.5*sin(2*pi*f(4)*t) + 2.5;
G_input_4 = laplace(desired_4);
y_f4(t) = ilaplace(G_input_4*G_plant);
tt = 0:0.001:1.9;
plot(tt, y_f4(tt), "--", "LineWidth", 2);
title("Response of Input Frequency of 15 Hz");
xlabel("Time (s)");
ylabel("Voltage (V)");
legend("Arduino Signal", "Desired Signal")
hold off

plot(Ass3_f20(:,1)/1e4, Ass3_f20(:,2), "LineWidth",2);
hold on

desired_4 = 2.5*sin(2*pi*f(6)*t) + 2.5;
G_input_4 = laplace(desired_4);
y_f4(t) = ilaplace(G_input_4*G_plant);
tt = 0:0.001:1.9;
plot(tt, y_f4(tt), "--", "LineWidth", 2);
title("Response of Input Frequency of 20 Hz");
xlabel("Time (s)");
ylabel("Voltage (V)");
legend("Arduino Signal", "Desired Signal")
hold off
```

```
plot(Ass3_f25(:,1)/1e4, Ass3_f25(:,2), "LineWidth",2);
hold on

desired_4 = 2.5*sin(2*pi*f(7)*t) + 2.5;
G_input_4 = laplace(desired_4);
y_f4(t) = ilaplace(G_input_4*G_plant);
tt = 0:0.001:1.9;
plot(tt, y_f4(tt), "--", "LineWidth", 2);
title("Response of Input Frequency of 25 Hz");
xlabel("Time (s)");
ylabel("Voltage (V)");
legend("Arduino Signal", "Desired Signal")
hold off



plot(Ass3_f30(:,1)/1e4, Ass3_f30(:,2), "LineWidth",2);
hold on

desired_4 = 2.5*sin(2*pi*f(8)*t) + 2.5;
G_input_4 = laplace(desired_4);
y_f4(t) = ilaplace(G_input_4*G_plant);
tt = 0:0.001:1.9;
plot(tt, y_f4(tt), "--", "LineWidth", 2);
title("Response of Input Frequency of 30 Hz");
xlabel("Time (s)");
ylabel("Voltage (V)");
legend("Arduino Signal", "Desired Signal")
hold off



plot(Ass3_f35(:,1)/1e4, Ass3_f35(:,2), "LineWidth",2);
hold on

desired_4 = 2.5*sin(2*pi*f(9)*t) + 2.5;
G_input_4 = laplace(desired_4);
y_f4(t) = ilaplace(G_input_4*G_plant);
tt = 0:0.001:1.9;
plot(tt, y_f4(tt), "--", "LineWidth", 2);
title("Response of Input Frequency of 35 Hz");
xlabel("Time (s)");
ylabel("Voltage (V)");
legend("Arduino Signal", "Desired Signal")
hold off



plot(Ass3_f40(:,1)/1e4, Ass3_f40(:,2), "LineWidth",2);
hold on

desired_4 = 2.5*sin(2*pi*f(10)*t) + 2.5;
G_input_4 = laplace(desired_4);
y_f4(t) = ilaplace(G_input_4*G_plant);
tt = 0:0.001:1.9;
plot(tt, y_f4(tt), "--", "LineWidth", 2);
title("Response of Input Frequency of 40 Hz");
```

```
xlabel("Time (s)");
ylabel("Voltage (V)");
legend("Arduino Signal", "Desired Signal")
hold off

clear s
s = tf('s');
G = 1/(1+122*1e-4*s);
[mag,ph,wout] = bode(G);
semilogx(2*pi*f,20*log10(Mag), "LineWidth", 2)
hold on
semilogx(wout,20*log10(squeeze(mag)),  "--", "LineWidth", 2);
hold off
title("Bode Diagram")
xlabel("Frequency")
ylabel("Magnitude")
legend("Experimental Result", "Theoric Result")
```