Controls Lab
Experiment 4
Noise cancellation in headphones

Monday, Table No. 12

Jayveer Singh Sikarwar: 21d070033
Sujeet Mehta: 21d070076
Bhavik Yadav: 21d070090

Contents

1 Objectives of the experiment

- To design and implement an analog circuit for noise cancellation in headphones.
- To achieve an attenuation of 20 dB(not necessarily exact 20, small attenuation also works if the system is stable) when a noise of 100 Hz frequency is applied.
- To design an analog compensator to stabilize the system, i.e., loop shaping of the loop transfer function.

2 Methodology and Design

We initially examined the headphone setup to understand its open-loop output transfer characteristics. Subsequently, we employed numpy, scipy, etc. libraries in Python to model the compensated stable system using a second-order approximation based on the gathered data. Following this, our attention turned to identifying suitable values of resistances and capacitors in the compensator, labeled as C(s).

The specific aim was to choose such a compensator that, at a frequency of 100 Hz, achieves a 20 dB(or similar) attenuation in the output-to-noise transfer function. It's crucial to emphasize that this analysis is centered on noise cancellation headphones, with a primary emphasis on enhancing the system's efficiency in minimizing undesired noise.
The flow diagram to observe open-loop characteristics is given below.

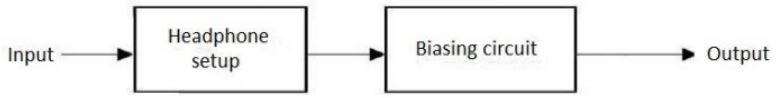

Fig 1: Open loop block diagram

Figure 1

This process highlights the practical implementation of engineering principles, focusing on elevating the headphone system's capacity to effectively reduce noise, especially at the specified frequency of concern. The use of Python libraries for system approximation demonstrates a refined strategy for data analysis and system optimization within the domain of experimental noise cancellation.

The closed-loop flow diagram is given below.

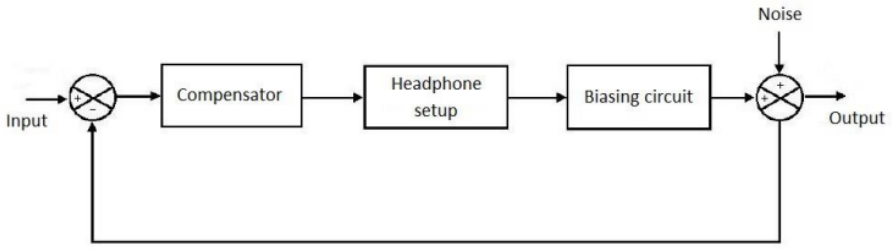

Fig 4: Closed loop block diagram

Figure 2

Now, we had to shift the gain at 100hz by 20db and make the closed compensated loop stable. For this, we had to use a compensator and find its correct resistances and capacitors. The compensator should be such that it provides a gain of 20db or so at 100hz. We used the second-order compensator as the first-order compensator was not able to provide both 20db gains and make our system stable by keeping the gain margin negative simultaneously.

2.1 Procedure

Frequency response analysis is performed on the headphone setup with sinusoidal waves given as input, from the function generator. Both the input and output are observed in the DSO, and then the magnitude and phase are plotted versus frequency. This is open open-loop characteristic of headphones.

2.1.1 Compensator Transfer function

Now, the compensator C(s) is of second order and is given by the equation

$$C(s) = \frac{as^2 + bs + c}{ds^2 + es + f}$$

Parameter	Value
a	0.006
b	0.006×5000
c	$0.006 \times 5000 \times 5000$
d	1
e	$250 + 300$
f	250×300

Table 1: Values for the above parameters.

This gives us the desired gain of approximately 20dB at 100 Hz for the closed-loop input-output transfer function.

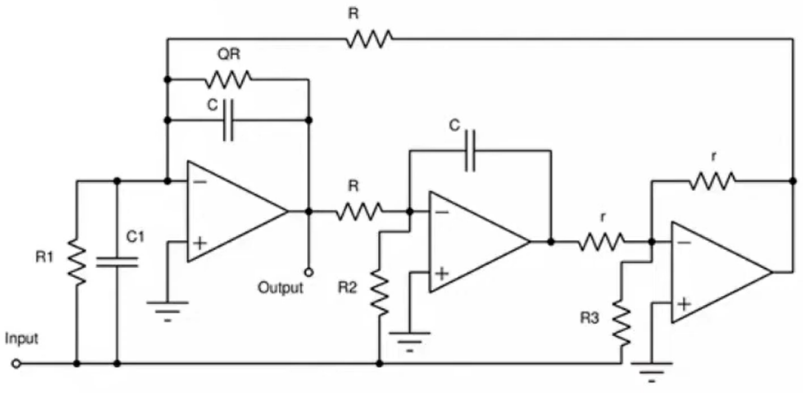

Figure 3: Bode plot of Compensator used

Parameter	Value
C	$10\,\mu\text{F}$
r	$10\,\Omega$
R	$365\,\Omega$
$R1$	$2500\,\Omega$
$R2$	$182.5\,\Omega$
$R3$	$300\,\Omega$
Q	0.5
$C1$	$775\,\text{nF}$

Table 2: Values for the given parameters.

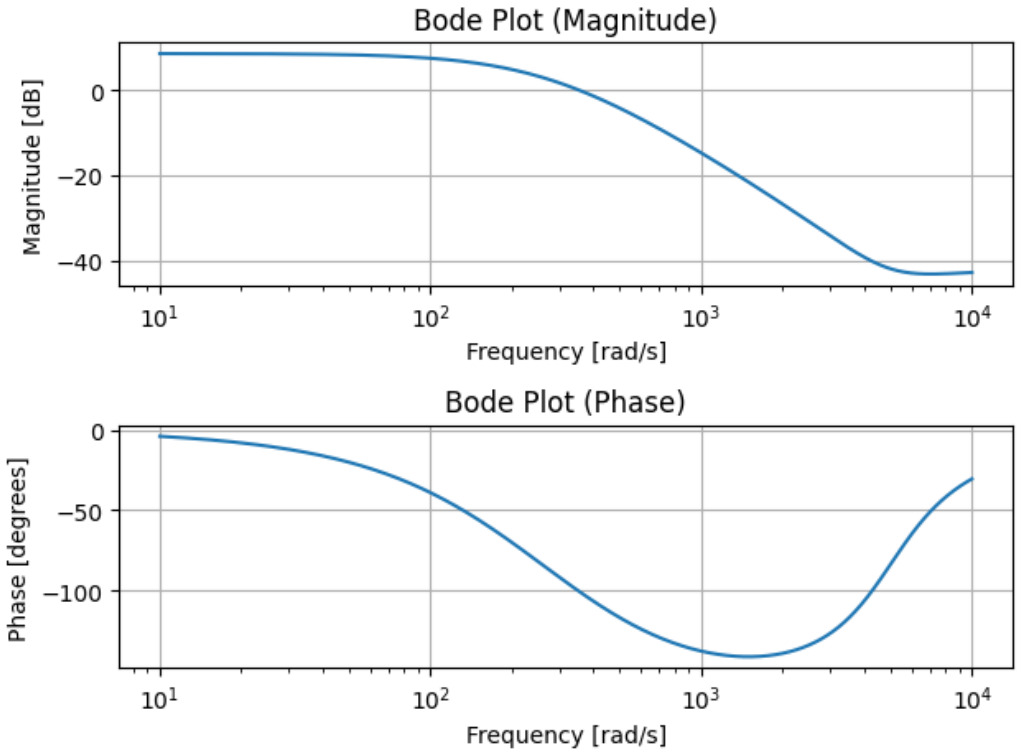

Figure 4: Bode plot of Compensator used

2.1.2 Uncompensated open-loop

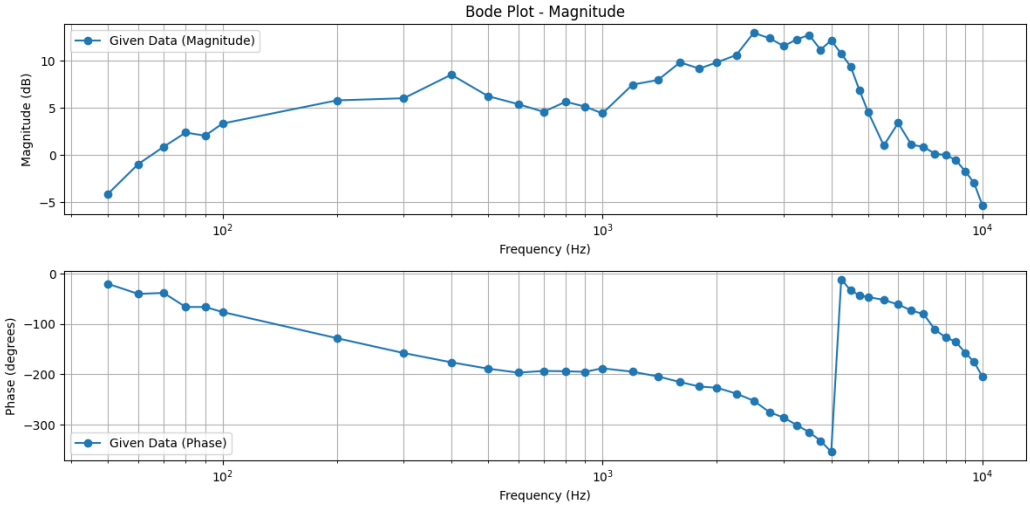

Figure 5: Uncompensated open-loop characteristic of headphones

2.1.3 Compensated open-loop

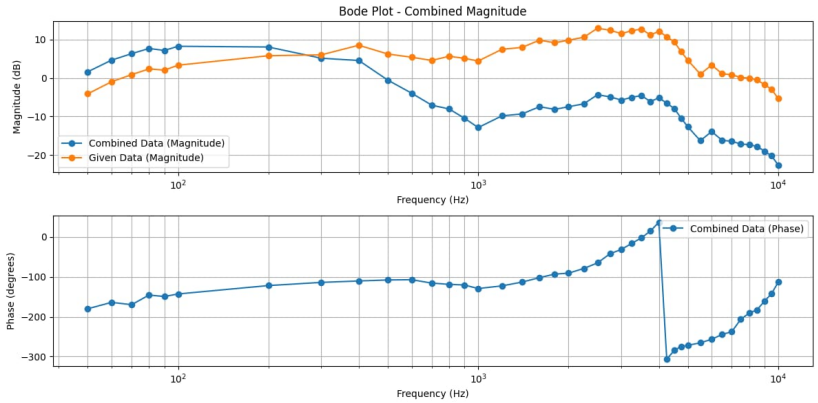

Figure 6: Compensated open loop bode plots.

2.1.4 Compensated closed-loop

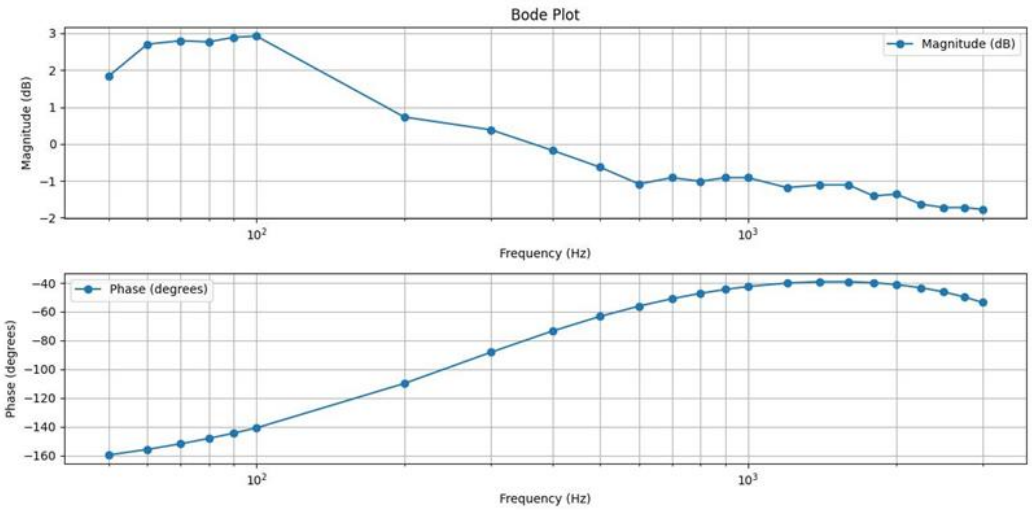

Figure 7: Compensated open loop bode plots.

2.2 Code

We used **Python** libraries for our plotting of data and whose codes are provided below-

2.2.1 Code for TF plotting

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import TransferFunction
from scipy import signal

# Define the values of R and C for the lag compensator

Kp_comp = 1.0  # Proportional gain
Ki_comp = 0.03  # Integral gain
Kd_comp = 0.0  # Derivative gain
```

```
Kp_obs = 1.0  # Observer Proportional gain
Ki_obs = 0.01  # Observer Integral gain
Kd_obs = 0.0  # Observer Derivative gain


num = [0.006, 0.006 * 5000, 0.006 * 5000 * 5000]
den = [1, 250 + 300, 250 * 300]
sys_compensator = TransferFunction(num, den)

# Frequency range for the Bode plot
w = np.logspace(1, 4, 1000)

# Calculate the magnitude and phase of the transfer function
w, mag, phase = signal.bode(sys, w)

# Create Bode plot
plt.figure()
plt.subplot(2, 1, 1)
plt.semilogx(w, mag)
plt.grid()
plt.xlabel('Frequency [rad/s]')
plt.ylabel('Magnitude [dB]')
plt.title('Bode Plot (Magnitude)')

plt.subplot(2, 1, 2)
plt.semilogx(w, phase)
plt.grid()
plt.xlabel('Frequency [rad/s]')
plt.ylabel('Phase [degrees]')
plt.title('Bode Plot (Phase)')

plt.tight_layout()
plt.show()
```

2.2.2 Code for uncompensated open loop bode plot

```
 import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import TransferFunction
from scipy import signal

# Given data
freq = np.array([50,60,70,80,90,100,200,300,400,500,600,700,800,900,1000,1200,
1400,1600,1800,2000,2250,2500,2750,3000,3250,3500,3750,4000,4250,4500,4750,5000,
5500,6000,6500,7000,7500,8000,8500,9000,9500,10000])  # Hz,
vout = np.array([860,1360,1680,1920,2000,2320,3040,3120,4160,3200,2680,2440,2760,
2600,2360,3400,3600,4400,4080,4400,4880,6400,6080,5520,5920,6320,5280,5840,
4960,240,3160,2440,1640,2040,1520,1480,1360,1360,1320,1200,1040,800])  # mV,
vin = np.array([1380,1520,1520,1460,1580,1580,1560,1560,1560,1560,1440,1440,
1440,1440,1420,1440,1440,1420,1420,1420,1440,1440,1460,1460,1440,1460,1460,
1440,1440,1440,1440,1440,1460,1380,1340,1340,1340,1360,1400,1460,1460,1480]) # mV
phase = np.array([-160,-140,-142,-114,-114,-104,-52,-22.4,-4.03,8.62,16.5,13.1,13.
7.9,14.7,24.1,35.2,44.2,46.5,58.2,72.3,95,106,121,135,152,174,-169,-147,-137,
-134,-128,-119,-107,-100,-69,-53.5,-45.3,-23.3,-5,24.5])  # degrees
phase = -180-phase


magnitude_dB = 20 * np.log10(np.array(vout) / np.array(vin))

w = np.logspace(2, 3, 1000)

w, mag_compensator, phase_compensator = signal.bode(sys, w)


plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)
plt.semilogx(freq, magnitude_dB, 'o-', label='Given Data (Magnitude)')
# plt.semilogx(w, mag_compensator, label='Lag Compensator (Magnitude)')
plt.title('Bode Plot - Magnitude')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude (dB)')
```

```
plt.grid(which='both', axis='both')
plt.legend()

plt.subplot(2, 1, 2)
plt.semilogx(freq, phase, 'o-', label='Given Data (Phase)')
# plt.semilogx(w, phase_compensator, label='Lag Compensator (Phase)')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Phase (degrees)')
plt.grid(which='both', axis='both')
plt.legend()

plt.tight_layout()
plt.show()
```

2.2.3 Code for Compensated open loop

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import TransferFunction
from scipy import signal
from scipy.interpolate import interp1d

# Given data
freq_given = np.array([50,60,70,80,90,100,200,300,400,500,600,700,800,900,1000,
1200,1400,1600,1800,2000,2250,2500,2750,3000,3250,3500,3750,4000,
4250,4500,4750,5000,5500,6000,6500,7000,7500,8000,8500,9000,9500,10000])  # Hz,
vout = np.array([860,1360,1680,1920,2000,2320,3040,3120,4160,3200,2680,2440,2760,
2600,2360,3400,3600,4400,4080,4400,4880,6400,6080,5520,5920,6320,5280,5840,
4960,4240,3160,2440,1640,2040,1520,1480,1360,1360,1320,1200,1040,800])  # mV,
vin = np.array([1380,1520,1520,1460,1580,1580,1560,1560,1560,1560,1440,1440,1440,
1440,1420,1440,1440,1420,1420,1420,1440,1440,1460,1460,1440,1460,1460,1440,
1440,1440,1440,1440,1460,1380,1340,1340,1340,1360,1400,1460,1460,1480])  # mV

phase_given = np.array([40,22,30,33,19.4,24,-36,-30,-15,-35,-20,-20,-70,-90,-102,-
-50,-40,-10,-4,-10,-7.5,-15])  # degrees output-input
phase_given = -180-phase
```

```
# Calculate magnitude in dB for given data
magnitude_dB = 20 * np.log10(vout / np.array(vin))

num = [0.006, 0.006 * 5000, 0.006 * 5000 * 5000]
den = [1, 250 + 300, 250 * 300]
sys_compensator = TransferFunction(num, den)

# Frequency range for the Bode plot
w = np.logspace(-2, 3, 1000)
w, mag_compensator, phase_compensator = signal.bode(sys_compensator, w)

mag_interpolated = np.interp(freq_given, w, mag_compensator)
phase_interpolated = np.interp(freq_given, w, phase_compensator)

# Add the magnitude and phase of the given data and the interpolated lag compensat
combined_magnitude = magnitude_dB + mag_interpolated
combined_phase = phase_given + phase_interpolated

# Create Bode plot for the combined data
plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)
plt.semilogx(freq_given, combined_magnitude, 'o-', label='Combined Data (Magnitude
plt.semilogx(freq_given, magnitude_dB, 'o-', label='Given Data (Magnitude)')
plt.title('Bode Plot - Combined Magnitude')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude (dB)')
plt.grid(which='both', axis='both')
plt.legend()

plt.subplot(2, 1, 2)
plt.semilogx(freq_given, combined_phase, 'o-', label='Combined Data (Phase)')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Phase (degrees)')
plt.grid(which='both', axis='both')
plt.legend()

plt.tight_layout()
```

```
plt.show()
```

2.2.4 Code for Compensated closed loop

```
import numpy as np
import matplotlib.pyplot as plt

# Given data
freq_given = np.array([50,60,70,80,90,100,200,300,400,500,600,700,800,900,1000,120
1400,1600,1800,2000,2250,2500,2750,3000])  # Hz,

vout = np.array([4940,5460,5520,5500,5580,5600,4480,4180,3920,3720,3600,
3600,3560,3600,3600,3560,3520,3520,3400,3420,3380,3280,3280,3260])  # mV,
vin = np.array([4000,4000,4000,4000,4000,4000,4120,4000,4000,4000,4080,
4000,4000,4000,4000,4080,4000,4000,4000,4000,4080,4000,4000,4000])  # mV

phase_given = np.array([-19.5,-15.4,-16.5,-13.8,-14,-14.5,-7.21,-2.17,-8.7,-0.007,
-5.2,-5.07,-5.76,1.94,2.9,-5.2,-5,-5.78,1.3,-1.45,-2.43,0,0,-2])
# degrees output-input
phase_given = -180-phase
#phase = phase -180.0
# Calculate magnitude in dB
magnitude_dB = 20 * np.log10(np.array(vout) / np.array(vin))

# Plot Magnitude (dB) vs. Frequency (Hz)
plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)
plt.semilogx(freq_given, magnitude_dB, 'o-', label='Magnitude (dB)')
plt.title('Bode Plot')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude (dB)')
plt.grid(which='both', axis='both')
plt.legend()

# Plot Phase (degrees) vs. Frequency (Hz)
plt.subplot(2, 1, 2)
plt.semilogx(freq_given, phase_given, 'o-', label='Phase (degrees)')
```

```
plt.xlabel('Frequency (Hz)')
plt.ylabel('Phase (degrees)')
plt.grid(which='both', axis='both')
plt.legend()

plt.tight_layout()
plt.show()
```

3 Challenges Faced

DIfference between simulation results and real results

Our journey had its tough moments. We were drawn to the idea of achieving perfection through simulations, but we knew we had to be cautious. Relying too much on virtual tools could be a problem. So, we realized the importance of testing our ideas in the real world. Moving from simulations to the actual headphone system required careful examination and adjustments. This part of the experiment showed us how crucial it is to connect what we expect from simulations with the unpredictable challenges of real-world performance.

Sentivity to the DSO/AFG

Every time we were taking the reading, there were some small errors occurring. The reading also depended on the AFG and DSO used. In the first lab, we took reading with someone else's DSO, and in the next class, we took with a new DSO and observed that there was a bit of change in readings which may affect our system. So we stick with the same AFG and DSO thereafter. Also for lower frequencies the output was coming correctly. It was taking time to stabilize the freq and the phase and for even lower freq like 50, 60hz, phase was varying too much.

Compensator tuning for component's values

Initially, we used a single lag compensator and applied the SISO tool in MATLAB to meet our specifications. However, we realized the need for two compensators—both lag and lead. After tuning them using the SISO tool, we achieved the desired compensator.

Facing challenges in noise cancellation became a driving force for innovation. Our experiment not only pushed the limits of compensator design but also highlighted the importance of a comprehensive and adaptable approach to solving complex engineering problems.

To tackle these challenges effectively, we adopted a holistic compensator design approach. This involved a combination of simulation-based tuning and real-world validation, taking into account factors like robustness and adaptability. This ensures that the compensator we design performs reliably across various operating conditions.