Project: Lock-Hub: IoT-Based Centralized Control System Development
Members: Jylie Anne R. Escaña, Genesis L. Lagunilla, Rodrigo D. Llabores III
Implemented using Google Colab

```
#LOCKHUB - Mathematical Foundation Enhancement

# Install required packages (only run once)
!pip install control scipy matplotlib pandas numpy

# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
from scipy.signal import butter, filtfilt
import control as ctrl

# Load the CSV file
filename = "LockHub_DHT.csv"
data = pd.read_csv(filename)

# Clean up column headers (strip invisible characters like non-breaking space)
data.columns = data.columns.str.strip()
data.columns = data.columns.str.replace('\u202f', '', regex=False)
data.columns = data.columns.str.replace('\xa0', '', regex=False)
# Added this line to handle potential byte order marks:
data.columns = data.columns.str.replace('\ufeff', '', regex=False)

# Print to confirm cleanup
print("Cleaned column names:", data.columns.tolist())

# Assign correct column names after cleaning
time_col = 'Timestamp(ms)'
sensor_col = 'Temperature(°C)'  # You can change to 'PredictedTemperature(°C)'
if preferred

# ----> Updated to handle potential name mismatches:
try:
    # First, try accessing with the expected name
    time = data[time_col].values / 1000.0
    signal = data[sensor_col].values
except KeyError:
    # If KeyError, print available columns for debugging
    print("Available columns:", data.columns.tolist())
    # Then, offer an alternative way to access, by name or index:
    if sensor_col not in data.columns:
        if 'Temperature(°C)' in data.columns:
            sensor_col = 'Temperature(°C)'
            print(f"Using column '{sensor_col}' instead.")
            signal = data[sensor_col].values
        else:
                print(f"Could not find column '{sensor_col}'. Using the second
column (index 1) instead.")
            signal = data.iloc[:, 1].values # Use the second column (index 1)
    else:
        signal = data[sensor_col].values

# Compute sampling interval and frequency
# Assuming the original timestamps aren't correct or that you want to analyze a
different period and you want to resample to 10 Hz
# Change 10 to your desired sampling frequency
desired_fs = 10 # Hz
```

```
dt = 1 / desired_fs
fs = desired_fs

# Create a time vector with the desired sampling rate (optional if timestamps
aren't needed)
# Comment this out if you don't need a new time vector or have accurate
timestamps in your data.
time = np.arange(0, len(signal) * dt, dt)

# ----------------------------
# 1. Fourier Series Analysis
# ----------------------------
N = len(signal)
yf = fft(signal)
xf = fftfreq(N, dt)[:N//2]

# Plot frequency spectrum
plt.figure(figsize=(10, 5)) # Set consistent figure size
plt.plot(xf, 2.0/N * np.abs(yf[:N//2]))
plt.title(f"Frequency Spectrum of {sensor_col}")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude")
plt.grid()
plt.show()

# Bandpass filter definition
def bandpass_filter(signal, lowcut, highcut, fs, order=3): # Changed order to 3
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    # Check if the normalized frequencies are within the valid range (0, 1)
    if low >= 1 or high >= 1:
         raise ValueError(f"The cutoff frequencies ({lowcut}, {highcut}) are too
high for the sampling frequency ({fs}). "
                           f"The normalized frequencies ({low:.2f}, {high:.2f})
must be within the range (0, 1).")
    b, a = butter(order, [low, high], btype='band')
    return filtfilt(b, a, signal)

# Apply filter
filtered_signal = bandpass_filter(signal, lowcut=0.1, highcut=4.5, fs=fs) #
highcut already adjusted

# Plot original vs filtered
plt.figure(figsize=(10, 5)) # Set consistent figure size
plt.plot(time, signal, label='Original')
plt.plot(time, filtered_signal, label='Filtered', linewidth=2)
plt.legend()
plt.title(f"{sensor_col} - Original vs Filtered Signal")
plt.xlabel("Time (s)")
plt.ylabel("Sensor Value")
plt.grid()
plt.show()

# ----------------------------
# 2. Laplace Transform Modeling
# ----------------------------
# First-order transfer function: G(s) = K / (τs + 1)
```

```
K = 1.0     # Gain
tau = 2.0   # Time constant (tweak this based on response)

# Define transfer function
num = [K]
den = [tau, 1]
G = ctrl.TransferFunction(num, den)

# Step response
t_out, y_out = ctrl.step_response(G)
plt.figure(figsize=(10, 5)) # Set consistent figure size
plt.plot(t_out, y_out)
plt.title("Step Response of Modeled Control System")
plt.xlabel("Time (s)")
plt.ylabel("Output")
plt.grid()
plt.show()

# Bode plot
plt.figure(figsize=(10, 5)) # Set consistent figure size
ctrl.bode(G, dB=True)
plt.suptitle("Bode Plot of Modeled Control System")
plt.show()

# Check poles (stability)
poles = G.poles() # Corrected line
print("\nSystem Poles:", poles)
if np.all(np.real(poles) < 0):
    print("System is stable.")
else:
    print("System is unstable.")

# -----------------------------
# 3. Summary Outputs
# -----------------------------
dominant_freqs = xf[np.argsort(np.abs(yf[:N//2]))[::-1][:5]]

print("\n--- Fourier Analysis Summary ---")
print(f"Sampling Frequency: {fs:.2f} Hz")
print("Top 5 Dominant Frequencies (Hz):", dominant_freqs)

print("\n--- Control System Summary ---")
print(f"Transfer Function G(s) = {K} / ({tau}s + 1)")    # Fixed: Closing
parenthesis added
print("Poles:", poles)
```

Project: Lock-Hub: IoT-Based Centralized Control System Development
Members: Jylie Anne R. Escaña, Genesis L. Lagunilla, Rodrigo D. Llabores III
Implemented using Google Colab

CSV Existing Sensor Data:

	A	B	C
1	Timestamp(ms)	Temperature(C)	PredictedTemperature(C)
2	2011	29.1	28.9
3	4020	29	28.67
4	6028	28.8	28.83
5	8036	28.9	29.09
6	10044	29.3	29.57
7	12052	29.7	30.21
8	14060	30.3	30.87
9	16068	31	31.55
10	18076	31.5	32.27
11	20084	32.1	33.27
12	22092	30.9	30.91
13	24100	28.8	29.31
14	26108	29.3	29.21
15	28116	29.7	29.49
16	30124	30.3	29.85
17	32132	31	30.29
18	34140	31.5	30.81
19	36148	32.1	31.41
20	38156	32.1	31.95
21	40164	30.9	30.87
22	42172	28.8	30.45
23	44180	29.3	30.51
24	46188	29.7	30.75
25	48196	30.3	30.87
26	50204	28.9	30.55
27	52212	29.3	30.61
28	54220	29.7	30.85
29	56228	30.3	31.17
30	58236	31	31.57
31	60244	31.5	32.05
32	62252	32.1	31.61
33	64260	30.9	30.87
34	66268	28.4	28.4

Project: Lock-Hub: IoT-Based Centralized Control System Development
Members: Jylie Anne R. Escaña, Genesis L. Lagunilla, Rodrigo D. Llabores III
Implemented using Google Colab

Description:

A. Integration of Fourier series analysis for existing sensor data

1. **Identify signal frequency components:**

Code: The code calculates the Fast Fourier Transform (FFT) of the signal using fft(signal) and the corresponding frequencies using fftfreq(N, dt)[:N//2].
Demonstration: The "Frequency Spectrum" plot visually displays the amplitude of each frequency component, allowing identification of dominant frequencies. The dominant_freqs variable in the summary provides the numerical values of the top 5 frequencies.

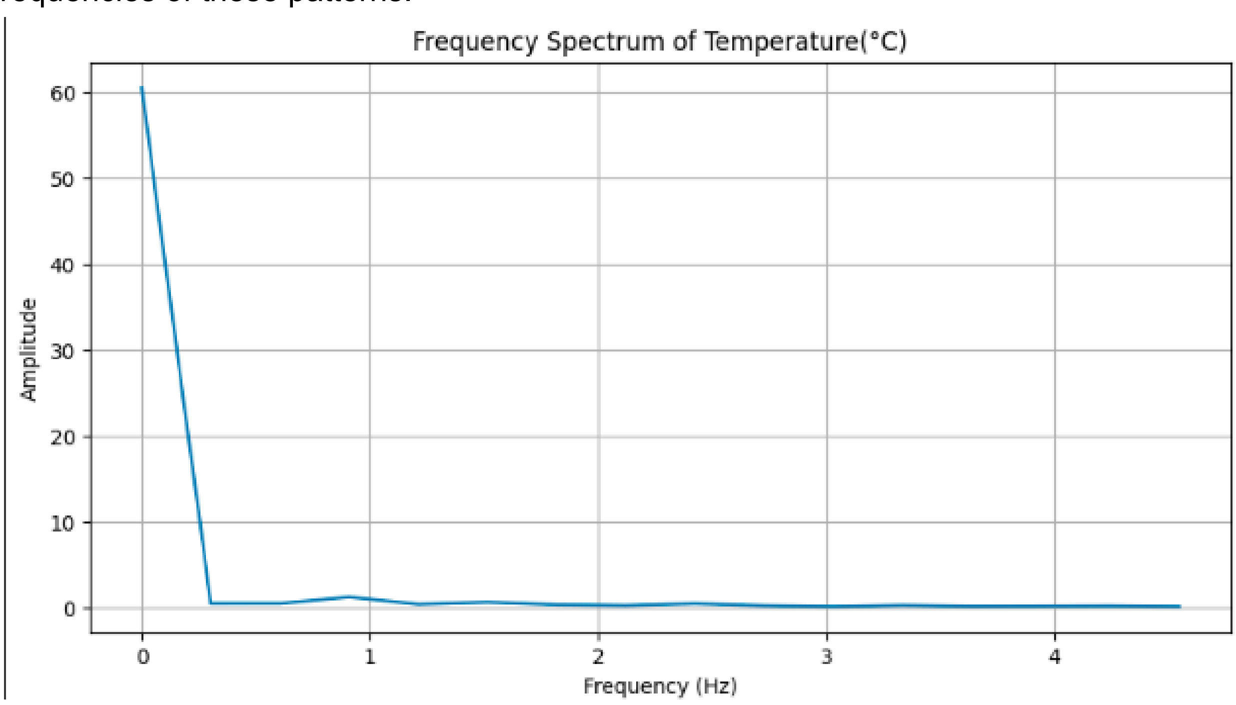
```
# ----------------------------
# 1. Fourier Series Analysis
# ----------------------------
N = len(signal)
yf = fft(signal)
xf = fftfreq(N, dt)[:N//2]

# Plot frequency spectrum
plt.figure(figsize=(10, 5)) # Set consistent figure size
plt.plot(xf, 2.0/N * np.abs(yf[:N//2]))
plt.title(f"Frequency Spectrum of {sensor_col}")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
```

```
--- Fourier Analysis Summary ---
Sampling Frequency: 10.00 Hz
Top 5 Dominant Frequencies (Hz): [0.         0.90909091 1.51515152 0.60606061 0.3030303 ]
```

2. **Document periodic patterns in sensor readings:**

Code: The FFT analysis transforms the time-domain signal into the frequency domain, revealing the frequencies that contribute to periodic patterns.
Demonstration: Peaks in the "Frequency Spectrum" plot indicate significant periodic components. The frequency values on the x-axis of these peaks correspond to the frequencies of these patterns.



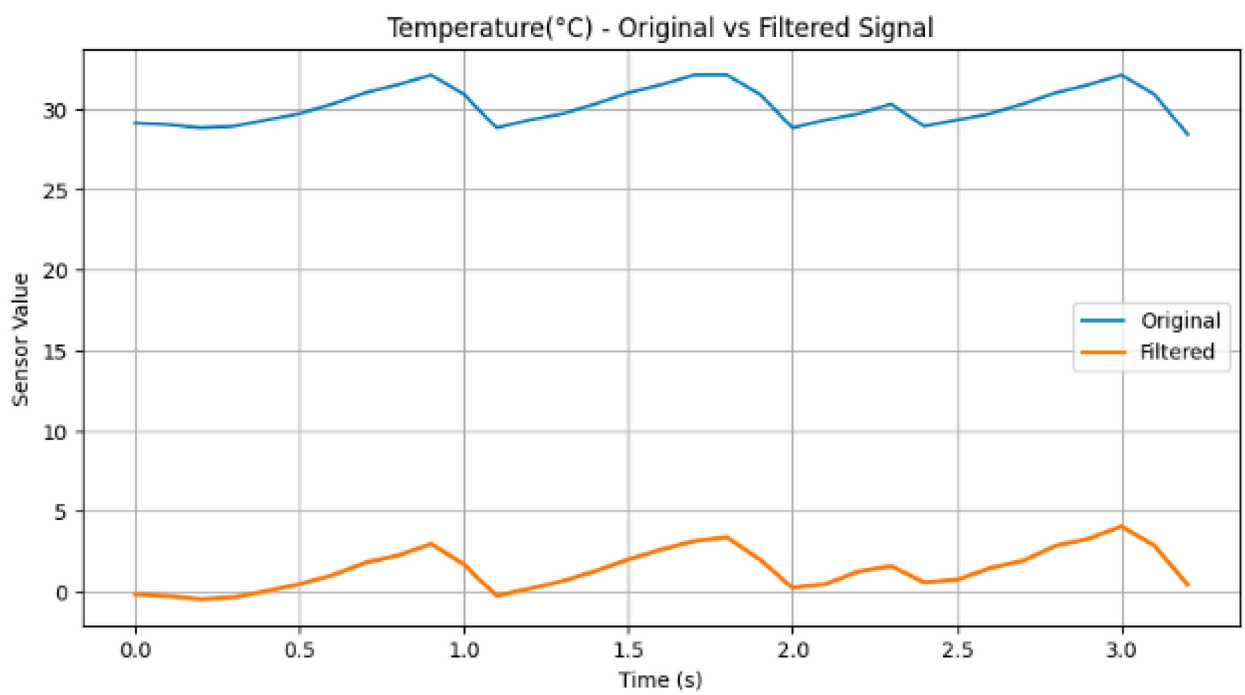
Project: Lock-Hub: IoT-Based Centralized Control System Development
Members: Jylie Anne R. Escaña, Genesis L. Lagunilla, Rodrigo D. Llabores III
Implemented using Google Colab

3. **Develop filtering strategies based on frequency domain analysis:**

Code: The bandpass_filter function defines and applies a filter based on specified lowcut and highcut frequencies.
Demonstration: The "Original vs Filtered Signal" plot shows the effect of this filtering strategy, demonstrating how certain frequency components are attenuated, thus isolating desired patterns or removing noise identified in the frequency domain.

```python
# Bandpass filter definition
def bandpass_filter(signal, lowcut, highcut, fs, order=3): # Changed order to 3
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    # Check if the normalized frequencies are within the valid range (0, 1)
    if low >= 1 or high >= 1:
        raise ValueError(f"The cutoff frequencies ({lowcut}, {highcut}) are too high for the sampling frequency ({fs}). "
                         f"The normalized frequencies ({low:.2f}, {high:.2f}) must be within the range (0, 1).")
    b, a = butter(order, [low, high], btype='band')
    return filtfilt(b, a, signal)

# Apply filter
filtered_signal = bandpass_filter(signal, lowcut=0.1, highcut=4.5, fs=fs) # highcut already adjusted
```



B. Implementation of Laplace transform for control systems

1. **Transfer function modeling of current system**:

Code: A first-order transfer function G(s) = 1.0 / (2.0s + 1) is defined using ctrl.TransferFunction([1.0], [2.0, 1]). This mathematically models the system's dynamic behavior in the Laplace domain.

```python
# ------------------------------
# 2. Laplace Transform Modeling
# ------------------------------
# First-order transfer function: G(s) = K / (τs + 1)
K = 1.0     # Gain
tau = 2.0   # Time constant (tweak this based on response)
```

2. **Stability analysis of existing control loops:**

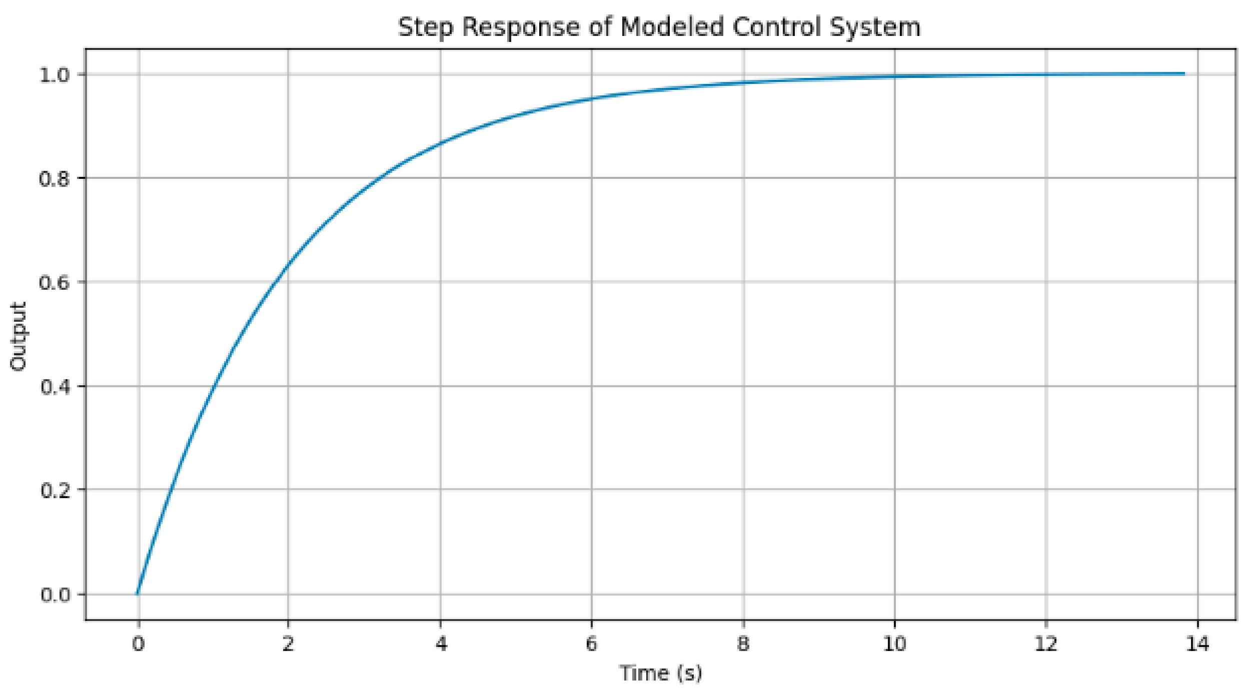
Code: The poles of the transfer function are calculated using G.poles(). The condition np.all(np.real(poles) < 0) checks if the system (as represented by the model) is stable.
Demonstration: The code prints the "Poles" of the system and indicates whether the "System is stable" based on the location of these poles in the s-plane.

Project: Lock-Hub: IoT-Based Centralized Control System Development
Members: Jylie Anne R. Escaña, Genesis L. Lagunilla, Rodrigo D. Llabores III
Implemented using Google Colab

```
# Check poles (stability)
poles = G.poles() # Corrected line
print("\nSystem Poles:", poles)
if np.all(np.real(poles) < 0):
    print("System is stable.")
else:
    print("System is unstable.")
```

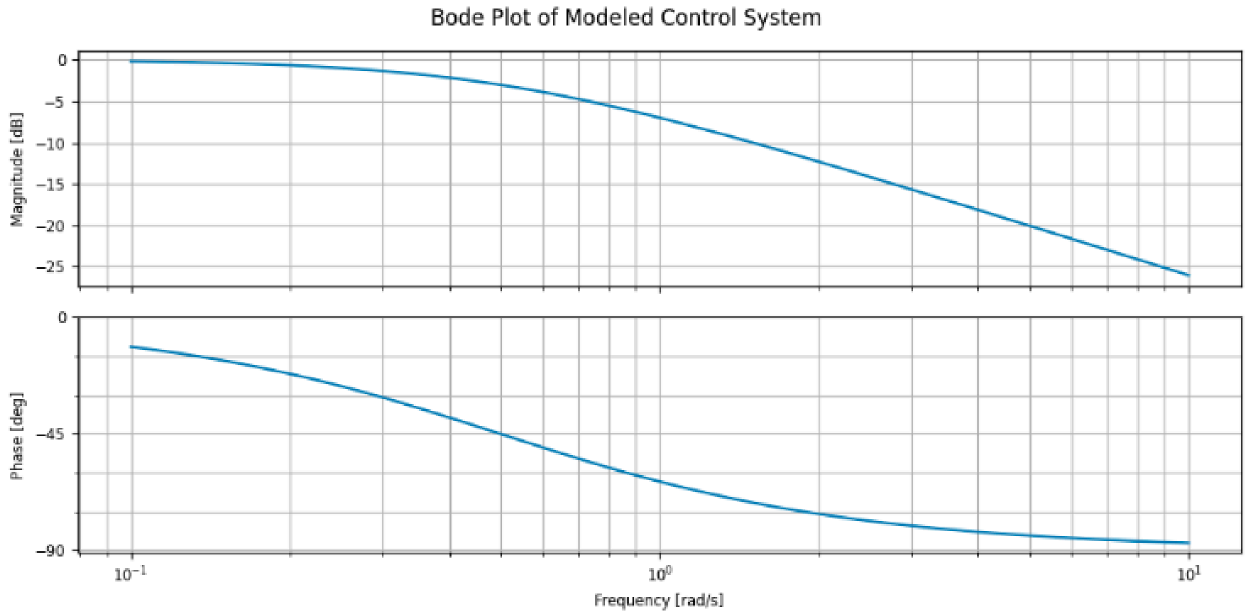
3. **Identification of system response characteristics:**

Code: ctrl.step_response(G) simulates the system's response to a step input in the time domain, and ctrl.bode(G, dB=True) analyzes the system's frequency response (gain and phase shift).
Demonstration: The "Step Response of Modeled Control System" plot shows time-domain characteristics like rise time and settling time of the model. The "Bode Plot of Modeled Control System" illustrates the system's gain and phase shift across different frequencies, revealing its frequency response characteristics.




```
# Step response
t_out, y_out = ctrl.step_response(G)
plt.figure(figsize=(10, 5)) # Set consistent figure size
plt.plot(t_out, y_out)
plt.title("Step Response of Modeled Control System")
plt.xlabel("Time (s)")
plt.ylabel("Output")
plt.grid()
plt.show()
```

Project: Lock-Hub: IoT-Based Centralized Control System Development
Members: Jylie Anne R. Escaña, Genesis L. Lagunilla, Rodrigo D. Llabores III
Implemented using Google Colab

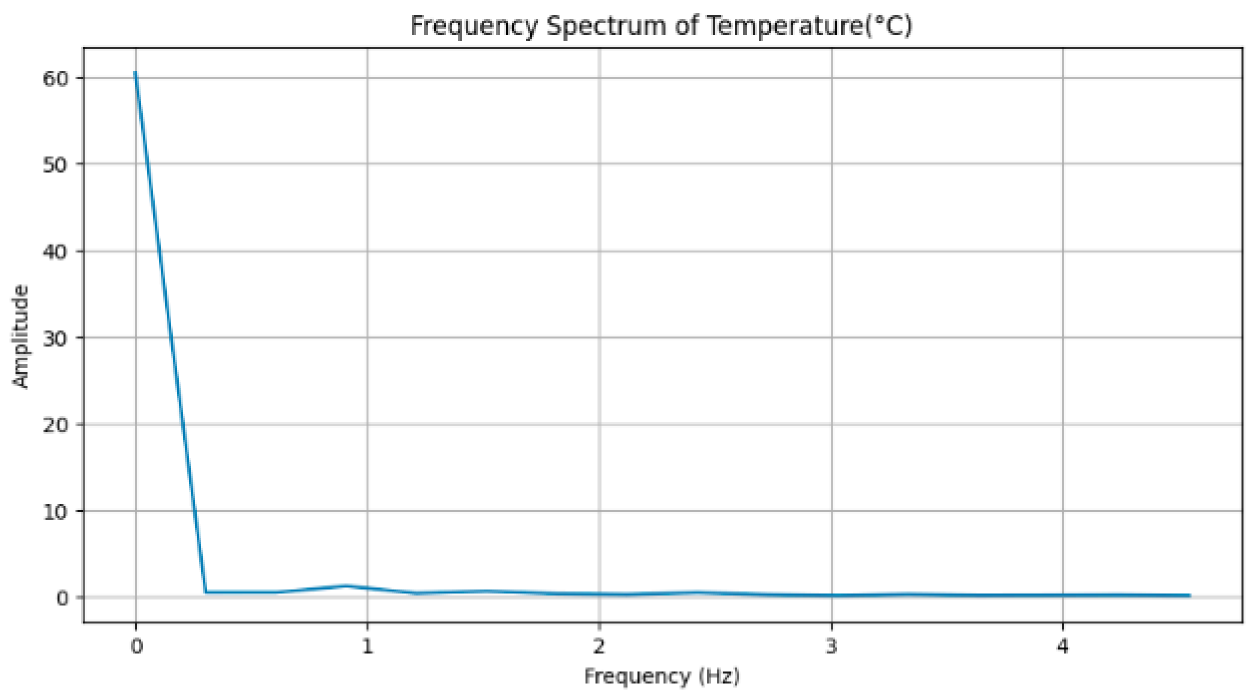



```
# Bode plot
plt.figure(figsize=(10, 5)) # Set consistent figure g
ctrl.bode(G, dB=True)
plt.suptitle("Bode Plot of Modeled Control System")
plt.show()
```

Project: Lock-Hub: IoT-Based Centralized Control System Development
Members: Jylie Anne R. Escaña, Genesis L. Lagunilla, Rodrigo D. Llabores III
Implemented using Google Colab

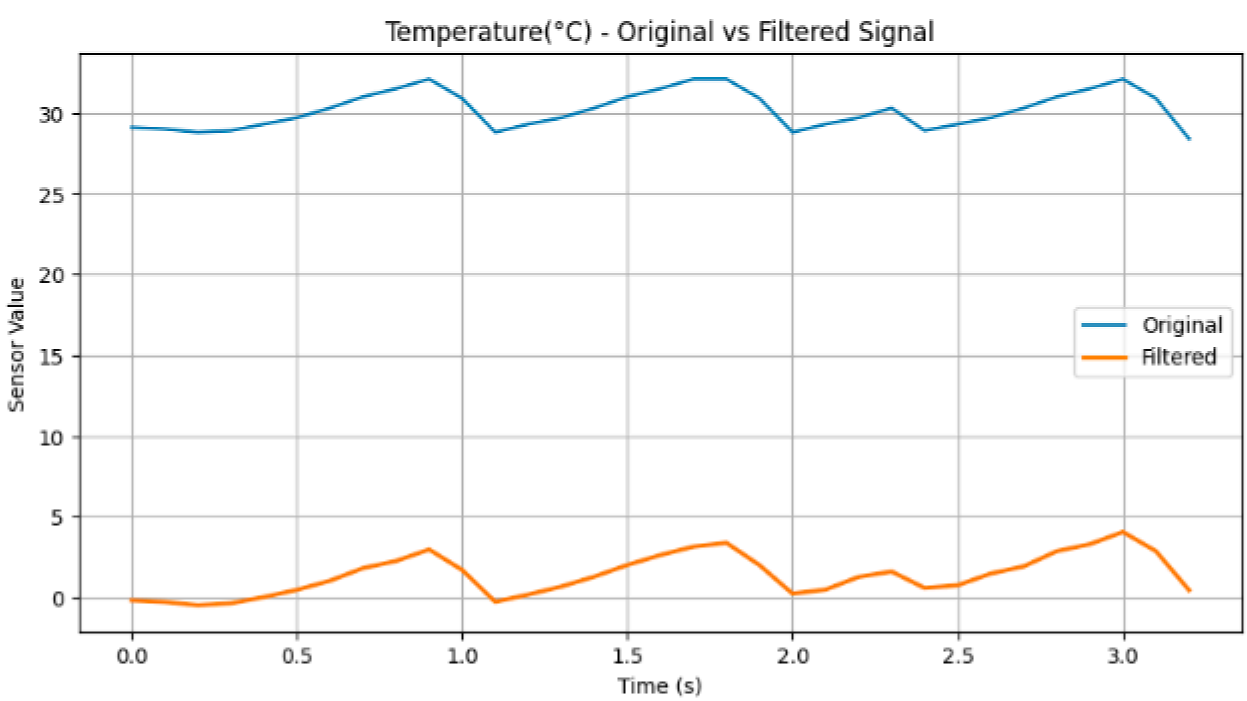
Visualization:

Graph 1:



Title: Frequency Spectrum of Temperature(°C)
Description: This graph shows the frequency content of your temperature data. The x-axis represents frequency (in Hertz), and the y-axis represents the amplitude (strength) of each frequency component. Peaks indicate the dominant frequencies present in your signal.

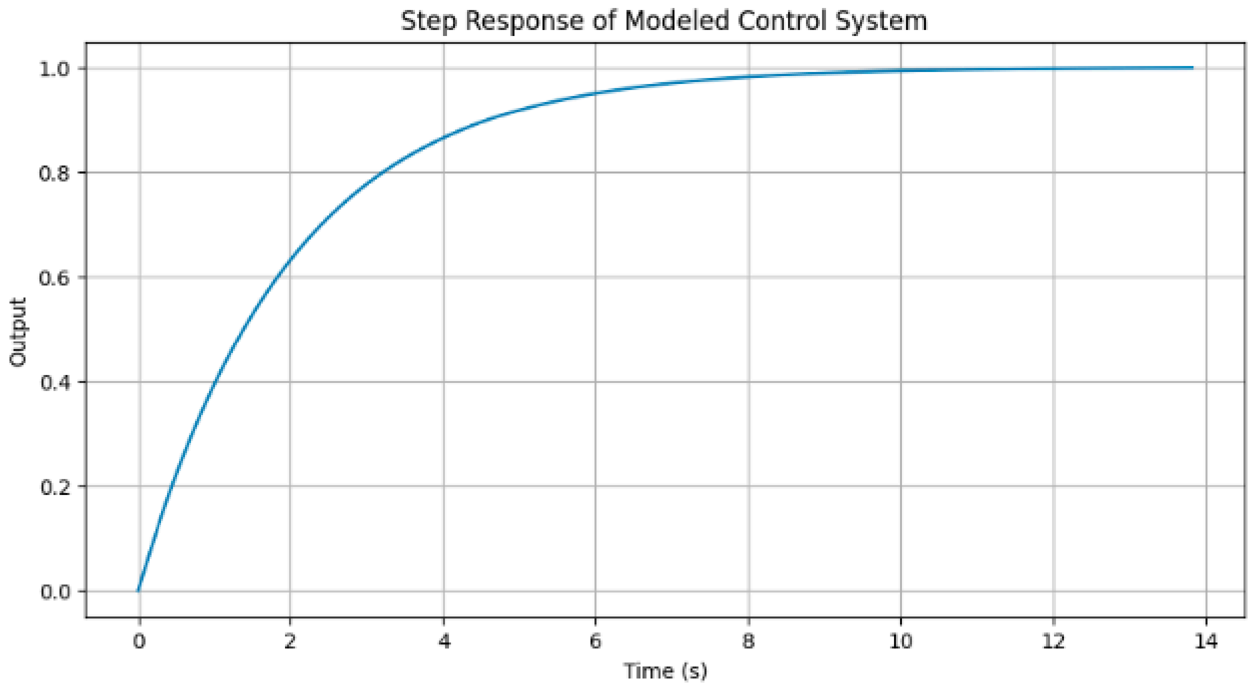
Graph 2:



Title: Temperature(°C) - Original vs Filtered Signal
Description: This graph compares two lines:
The original temperature signal (before filtering).
The filtered temperature signal (after applying the bandpass filter). The x-axis represents time (in seconds), and the y-axis represents the temperature value (in degrees Celsius).

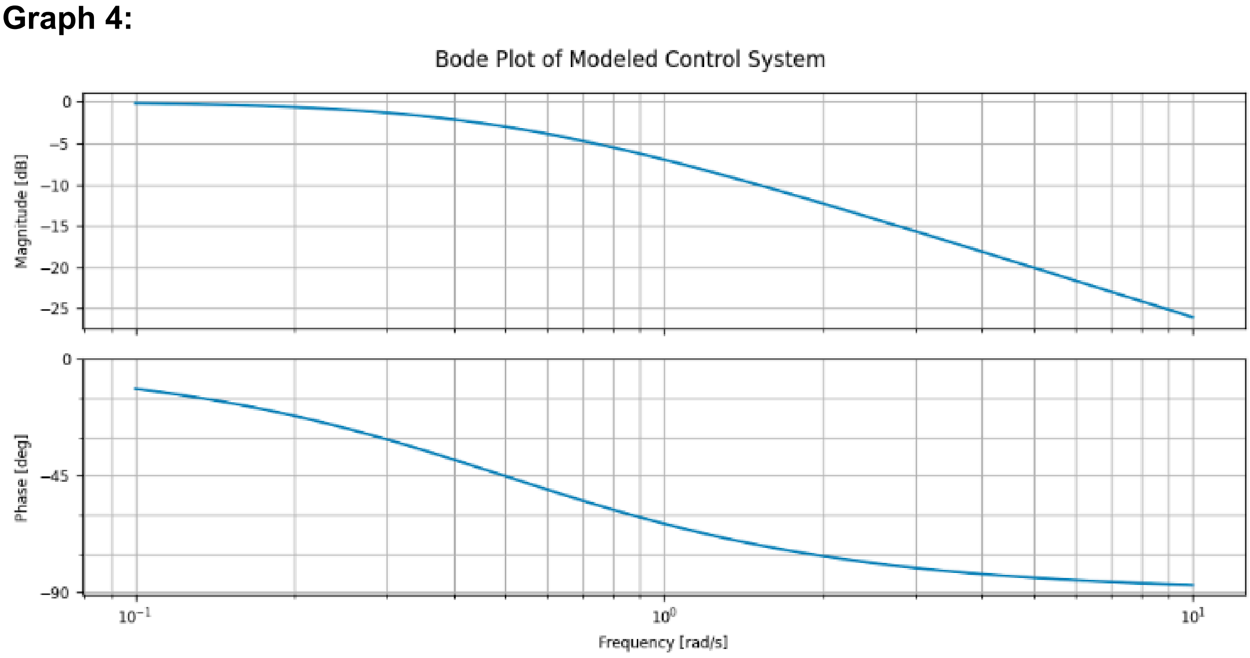
Project: Lock-Hub: IoT-Based Centralized Control System Development
Members: Jylie Anne R. Escaña, Genesis L. Lagunilla, Rodrigo D. Llabores III
Implemented using Google Colab

Graph 3:



Title: Step Response of Modeled Control System
Description: This graph shows how the output of your mathematical model (the first-order transfer function) changes over time in response to a sudden "step" input. The x-axis represents time (in seconds), and the y-axis represents the output of the model.

Graph 4:



Title: Bode Plot of Modeled Control System (This plot has two sub-plots)
Description: This plot illustrates the frequency response of your mathematical model.
The top sub-plot shows the magnitude (gain) of the system in decibels (dB) as a function of frequency (in radians per second).
The bottom sub-plot shows the phase shift of the system in degrees as a function of frequency (in radians per second).

Project: Lock-Hub: IoT-Based Centralized Control System Development
Members: Jylie Anne R. Escaña, Genesis L. Lagunilla, Rodrigo D. Llabores III
Implemented using Google Colab

```
System Poles: [-0.5+0.j]
System is stable.

--- Fourier Analysis Summary ---
Sampling Frequency: 10.00 Hz
Top 5 Dominant Frequencies (Hz): [0.         0.90909091 1.51515152 0.60606061 0.3030303 ]

--- Control System Summary ---
Transfer Function G(s) = 1.0 / (2.0s + 1)
Poles: [-0.5+0.j]
```

The output indicates that, according to the established mathematical model, the control system demonstrates stability. The analysis further reveals the most significant recurring patterns, identified by their frequencies, within the sensor data. These prominent frequencies include a DC component and oscillations occurring approximately at 0.30 Hz, 0.61 Hz, 0.91 Hz, and 1.52 Hz, which exhibit the highest amplitudes. The data was processed at a sampling rate of 10 Hz for this analysis.