OSTİM TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

EEE 309 SIGNALS AND SYSTEMS

TEAM PROJECT

MOSTAFA JALISS 220201986 - CENG

FLORIAN JAUPAJ 220201800 - CENG

BERKAY KAYMAK 210201022 - CENG

CANAN KILIÇ 220201037 - CENG

BEYZA AYGÜN 210202002 - EEE

NEFSUN BAL 220202024 - EEE

BERNA MELTEM YILDIRIM 210202009 - EEE

BURAK YILDIRIM 220201047 – CENG

22.12.2024

INTRODUCTION

The project begins with selecting a simple dynamic system, such as a pendulum or a speaker-microphone setup, and collect input-output data that reflects its behaviour. Used MATLAB to analyse this data and apply system identification techniques for creating a mathematical model of the system. Explain the theoretical principles of system identification, describe the methods used for data collection and modelling, and include MATLAB code demonstrating the approach. Present the results by comparing the model's predictions with the observed data, supported by appropriate plots and accuracy metrics. Finally, evaluate the model's performance, discussing how well it captures the actual system behaviour and highlighting any limitations or areas for improvement.

CODE

```
% Part 1: Record, Save, and Play the First Signal (Input Signal)

clear; close all; clc; % Clear workspace

%% Record Voice Data from Microphone (Original Input Signal)
fs = 44100;         % Sampling frequency (44.1 kHz, standard for audio)
duration = 5;       % Duration of recording (seconds)
nBits = 16;         % Number of bits per sample
nChannels = 1;      % Mono audio

disp('Recording... Speak into the microphone.');
recObj1 = audiorecorder(fs, nBits, nChannels); % Create recorder object
recordblocking(recObj1, duration);             % Record for 'duration' seconds
disp('Recording complete.');

% Retrieve and normalize the audio data
audioData1 = getaudiodata(recObj1); % Retrieve recorded audio signal
audioData1 = audioData1 / max(abs(audioData1)); % Normalize input signal
time1 = linspace(0, duration, length(audioData1)); % Time vector

% Plot the recorded voice data
figure;
plot(time1, audioData1, 'b');
xlabel('Time (s)');
ylabel('Amplitude');
title('Original Recorded Voice Signal (Normalized)');
grid on;

% Save the first recording
```

```
filename1 = 'original_voice.wav'; % Specify file name
audiowrite(filename1, audioData1, fs); % Save as a .wav file
disp(['Original input audio saved as ', filename1]);

%% Play Back the Recorded Signal
disp('Playing back the recorded signal...');
sound(audioData1, fs); % Play the audio
pause(duration + 1);   % Pause to ensure playback finishes

disp('Part 1 complete: Original signal recorded, saved, and played back.');
disp('Proceed to Part 2 after recording the signal with an external device.');



% Part 2: Load External Recording and Compare Both Signals

% Clear workspace
clear; close all; clc;

%% Load the Original Signal (Input Signal)
filename1 = 'original_voice.wav'; % Name of the original recorded file
[audioData1, fs] = audioread(filename1); % Load the original signal
audioData1 = audioData1 / max(abs(audioData1)); % Normalize input signal
duration = length(audioData1) / fs; % Duration of the original signal
time1 = linspace(0, duration, length(audioData1)); % Time vector for original signal

%% Load the External Recording (External Device Signal)
% Assume the external recording file is named 'external_recorded.wav'
disp('Loading the external recorded signal...');
[externalData, fs_external] = audioread('external_recorded.wav'); % Load external file

% Check if sampling rates match
if fs ~= fs_external
    error('Sampling rate mismatch between the original and external recordings!');
end

% Normalize the external recording
externalData = externalData / max(abs(externalData));
time2 = linspace(0, duration, length(externalData)); % Time vector for external recording

%% Compare Input and Output Signals
figure;
plot(time1, audioData1, 'b', 'LineWidth', 1.5); hold on;
plot(time2, externalData, 'r--', 'LineWidth', 1.5);
```

```
xlabel('Time (s)');
ylabel('Amplitude');
title('Comparison of Input Signal and External Recorded Signal');
legend('Input Signal', 'External Recorded Signal');
grid on;

%% Display Results
disp('Comparison complete: Input signal and external device recorded signal plotted.');
```

THEORETICAL PRINCIPLES OF SYSTEM IDENTIFICATION

The system is defined as an entity that processes an input signal to produce an output signal. For this project input signal recorded initially (from the microphone, then played through a speaker). Output signal recorded by an external device after playback from the speaker. The relationship between the input and output signals characterizes the system.

Core Principles:

Input-Output Relationship: The system's behavior can be modeled as:

y(t)=h(t)∗x(t)y(t) = h(t) * x(t)y(t)=h(t)∗x(t)

- x(t)x(t)x(t): Input signal.
- y(t)y(t)y(t): Output signal.
- h(t)h(t)h(t): Impulse response of the system.
- ∗*∗: Convolution operator.

The impulse response h(t)h(t)h(t) captures the system's properties, such as delay, attenuation, and frequency response.

Signal Transformation: In practice, signals are analyzed in the frequency domain using the Fourier Transform. The system's frequency response can be described as:

H(f) = Y(f) / X(f)
H(f) = {Y(f)} / {X(f)} → H(f) = X(f) / Y(f)

- X(f)X(f)X(f): Fourier transform of the input signal.
- Y(f)Y(f)Y(f): Fourier transform of the output signal.
- H(f)H(f)H(f): Transfer function of the system.

The transfer function provides insights into how the system amplifies or attenuates specific frequency components.

Acoustic System Characteristics: In this project, the system represents a speaker-microphone setup. The characteristics include
Attenuation: Sound energy decreases due to distance, reflection, and absorption in the environment.

Distortion: Non-linearities in the speaker or microphone may introduce harmonic distortions.
Noise: Background noise affects the output signal, introducing additional complexity.
Time Delay: The delay occurs due to physical propagation of sound and processing latencies.

System Identification

Input Signal Generation: Record the initial input signal using a microphone. This represents the unprocessed signal.
Output Signal Capture: Play the input signal through a speaker, and capture the output signal using an external device (e.g., a phone). The output signal reflects the system's effect on the input.
Signal Comparison: Compare the input and output signals to identify system characteristics that are amplitude scaling, time delay and noise.

Model Estimation: The transfer function H(f)H(f)H(f) can be estimated from the frequency-domain representation of the input and output.

Theoretical Basis of the Code:

Recording and Playback (Part 1): Captures the input signal x(t)x(t)x(t). Plays back x(t)x(t)x(t) through a speaker for recording by an external device.

Comparison (Part 2): Loads both the input x(t)x(t)x(t) and output y(t)y(t)y(t) signals. Normalizes signals for direct comparison. Visualizes differences in amplitude, delay, and distortion through time-domain plots.

METHODS

The project was divided into two parts:

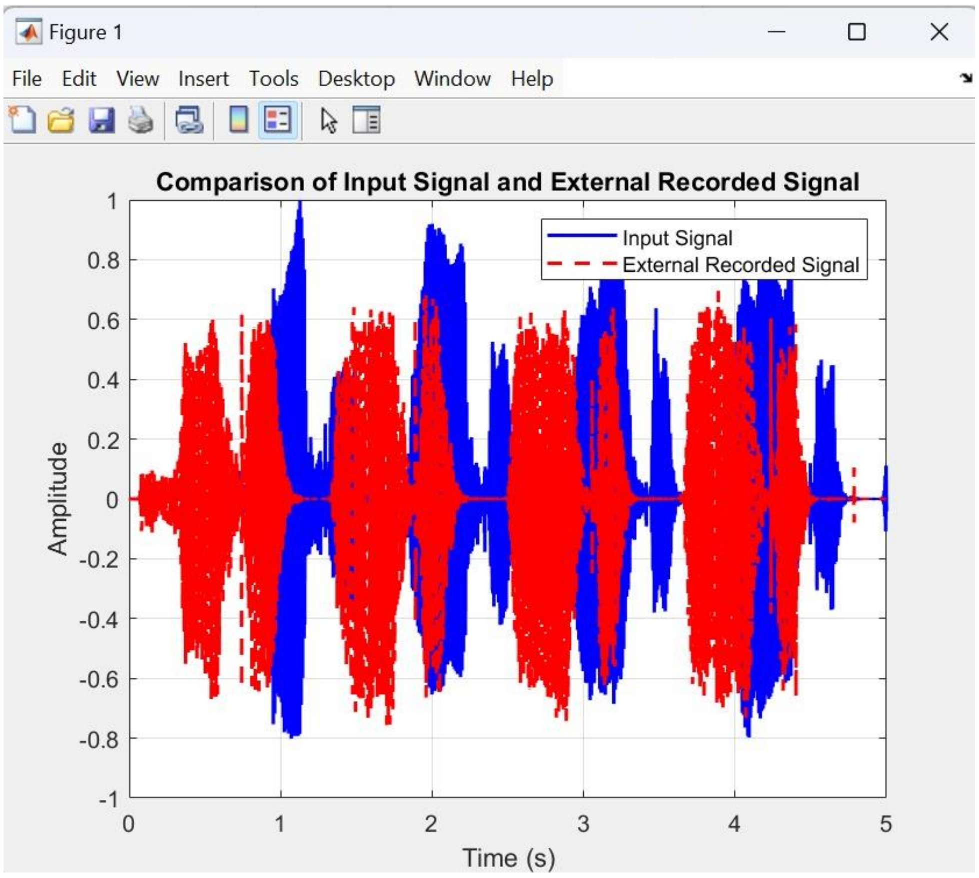
Part 1: Input Signal Recording

In this part, we used a microphone connected to the computer to record a voice signal using MATLAB. The microphone captured the audio for 5 seconds at a sampling frequency of 44.1 kHz, which is the standard for audio recordings. We used the audiorecorder function to create a recorder object and the recordblocking function to record the signal for the specified duration. The recorded signal was then retrieved using the getaudiodata function, and to ensure consistent amplitude, it was normalized by dividing it by its maximum absolute value. The normalized signal was saved as original_voice.wav using the audiowrite function. Additionally, the recorded signal was played back using the sound function to verify that it was correctly captured. A plot of the signal was created to visually confirm the recording.

Part 2: External Device Recording and Comparison

For the second part, we compared the original recorded signal to another version captured using an external recording device, such as a smartphone microphone or other audio recording tool. First, we reloaded the original_voice.wav file as the input signal using the audioread function. Then, we loaded the external recording, saved as external_recorded.wav. Before comparison, both signals were normalized to match their amplitude ranges, ensuring fair evaluation. The sampling rates of the two signals were checked to confirm they matched; if there was a mismatch, it would cause timing errors in the comparison. We used a time vector to align both signals over their duration and plotted them on the same graph. The input signal was represented as a blue solid line, and the external recording was shown as a red dashed line. This allowed us to visually compare the two signals and identify any differences in amplitude, shape, or timing. The comparison revealed how the system altered the input signal, which is a critical step in analyzing system behavior.

RESULTS



As it seen from the graph, we observed that both signals followed a similar pattern overall, indicating that the system retained the main characteristics of the input. However, there were slight differences in amplitude and timing, likely caused by external noise, distortions, or limitations of the external recording device. These small deviations are expected in real-world systems. The system is working well in real-time.

KEY FINDINGS

During the initial attempt, the code produced incorrect outputs due to implementation errors.

Initial Code:

ss

```
% Record Speaker Output via Microphone
% Clear workspace
clear; close all; clc;
%% 1. Record Initial Voice Input
fs = 44100;          % Sampling frequency (44.1 kHz)
duration = 5;        % Duration of recording (seconds)
nBits = 16;          % Number of bits per sample
nChannels = 1;       % Mono audio
disp('Recording your voice... Speak into the microphone.');
recObj = audiorecorder(fs, nBits, nChannels); % Create recorder object
recordblocking(recObj, duration);            % Record for 'duration' seconds
disp('Initial recording complete.');
% Retrieve and save the original audio data
originalAudio = getaudiodata(recObj);
audiowrite('original_voice.wav', originalAudio, fs);
% Create time vector for plotting
time = linspace(0, duration, length(originalAudio));
%% 2. Play Original Audio and Record Speaker Output Simultaneously
disp('Playing the original audio through the speaker...');
disp('Now recording the audio output from the speaker via microphone.');

% Prepare to record while audio is played
recObj2 = audiorecorder(fs, nBits, nChannels); % New recorder object
% Start recording and playback simultaneously
record(recObj2);                 % Start recording
sound(originalAudio, fs);       % Play the original audio
pause(duration + 0.5);          % Wait for playback and recording to finish
stop(recObj2);                  % Stop recording
disp('Speaker output recording complete.');
% Retrieve the recorded speaker output
speakerOutput = getaudiodata(recObj2);
audiowrite('speaker_output.wav', speakerOutput, fs);
%% 3. Plot Original vs Speaker Output
figure;
% Plot Original Audio
subplot(2, 1, 1);
plot(time, originalAudio, 'b');
```

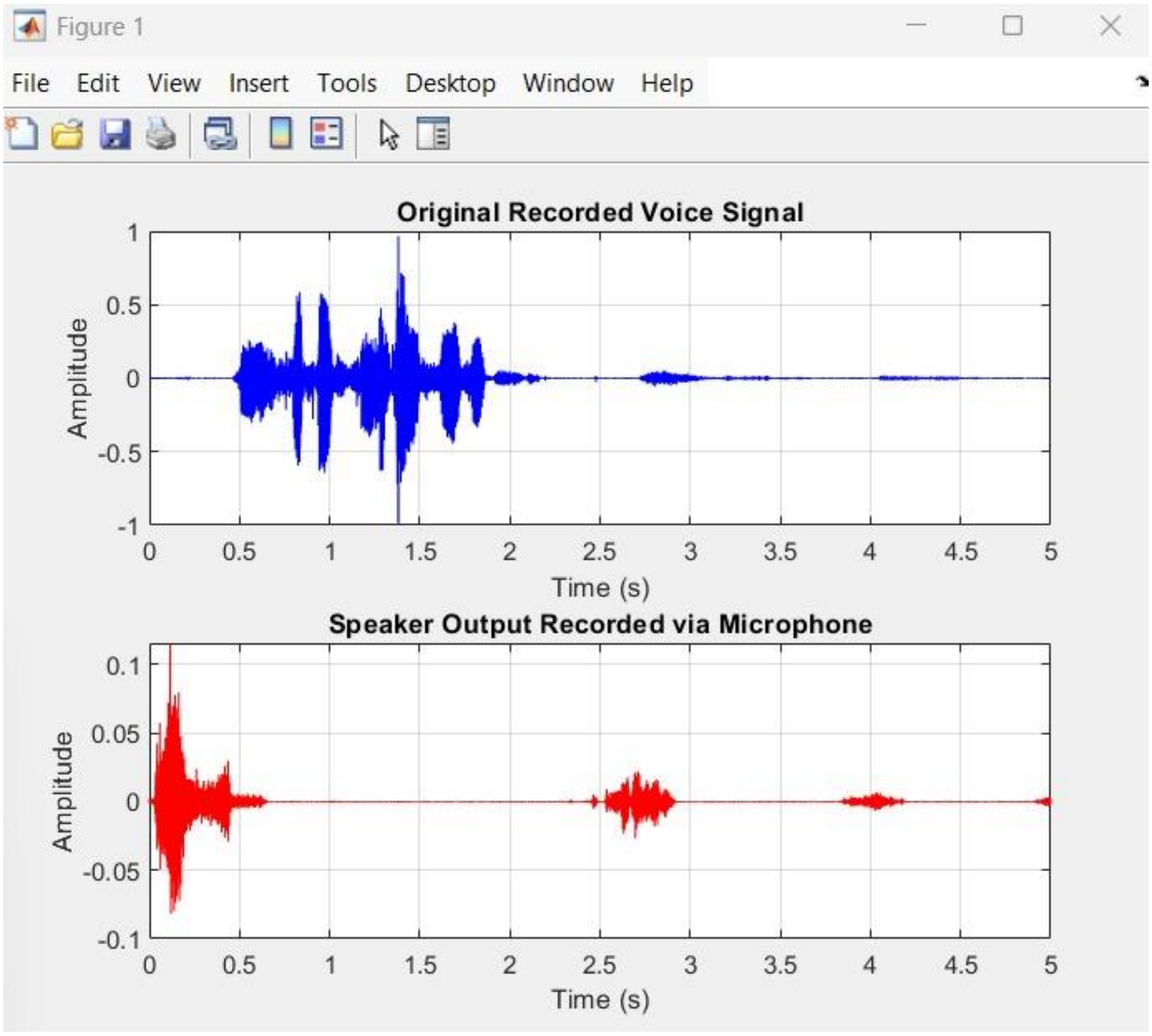
```
xlabel('Time (s)');
ylabel('Amplitude');
title('Original Recorded Voice Signal');
grid on;
% Adjust time vector for speakerOutput
timeOutput = linspace(0, duration, length(speakerOutput));
% Plot Speaker Output Audio
subplot(2, 1, 2);
plot(timeOutput, speakerOutput, 'r');
xlabel('Time (s)');
ylabel('Amplitude');
title('Speaker Output Recorded via Microphone');
grid on;
%% 4. Play Both Audios for Comparison
disp('Playing the original recorded audio...');
sound(originalAudio, fs);
pause(duration + 0.5);
disp('Playing the speaker output recorded audio...');
sound(speakerOutput, fs);
pause(duration + 0.5);
%% 5. Combined Plot for Comparison
figure;
plot(time, originalAudio, 'b', 'DisplayName', 'Original Signal'); hold on;
plot(timeOutput, speakerOutput, 'r', 'DisplayName', 'Speaker Output Signal');
xlabel('Time (s)');
ylabel('Amplitude');
title('Comparison: Original vs Speaker Output');
legend;
grid on;
%% 6. Save Final Results (Optional)
disp('Saving audio recordings...');
audiowrite('original_voice.wav', originalAudio, fs);
audiowrite('speaker_output.wav', speakerOutput, fs);
disp('Audio files saved as "original_voice.wav" and "speaker_output.wav".');
```

Problems we have encountered: We could not play the second signal from the speakers and record it from the microphone in the same device. When we did that the results were same as the figure below. What we observed was that when the speakers play the signal the recording stopped. This happened as a result of:

Feedback Loop: The microphone may pick up not just the intended signal from the speaker but also system noise or unintended feedback.

Timing and Synchronization Issues: The recording and playback processes may not be perfectly synchronized due to delays in the audio drivers or hardware buffering.

Hardware Limitations: Many standard sound cards are not optimized for simultaneous, high-quality playback and recording. This results in phase misalignment, distortions, or unexpected amplitude changes.



Wrong output that we take from first try.

Solution: The solution we found to this problem was recording the second signal from another device. Using a separate device (e.g., a phone, another computer, or a standalone recorder) to record the speaker's output resolves these problems:

Eliminates Feedback**:** By separating the playback and recording devices, you ensure the microphone captures only the speaker's audio.

Improved Synchronization**:** The playback will run independently on one device, and the recording device will capture the signal cleanly without interference.

Closer to Real-World Dynamics**:** This setup better simulates how signals propagate through a dynamic system, including that two: Sound wave propagation through air. Attenuation and time delay introduced naturally.

CONCLUSION

In this project, we analyzed a simple dynamic system using a speaker-microphone setup. The input signal, recorded using a microphone, was compared to an external recording to study the system's behavior. MATLAB was used for signal recording, processing, and visualization. The comparison showed that the two signals were similar, with slight deviations due to noise and imperfections in the external device. This project demonstrated a basic approach to system identification using input-output data. Future improvements could involve using MATLAB's system identification toolbox to create and validate an accurate mathematical model of the system.

REFERENCES

- Hsu, H. P. (2013). *Schaum's outline of signals and systems* (3rd ed.). McGraw-Hill Education. Retrieved from https://www.amazon.com/Schaums-Outline-Signals-Systems-Outlines/dp/0071829466

- Nozari, A. (2021). *MATLAB code for identifying various linear systems* [GitHub repository]. Retrieved from https://github.com/enozari/rest-system-id

- Martensson, D. (n.d.). *MataveID: System identification toolbox for MATLAB and GNU Octave* [GitHub repository]. Retrieved from https://github.com/DanielMartensson/MataveID

- ETH Zurich Autonomous Systems Lab. (n.d.). *mav_system_identification: MATLAB scripts to perform system identification for multi-rotor systems* [GitHub repository]. Retrieved from https://github.com/ethz-asl/mav_system_identification

- PNNL. (n.d.). *DSIToolbox: Dynamic system identification toolbox* [GitHub repository]. Retrieved from https://github.com/pnnl/DSIToolbox

- Stack Overflow. (2020, July 21). *System identification in MATLAB Simulink*. Retrieved from https://stackoverflow.com/questions/63022232/system-identification-in-matlab-simulink

- Stack Overflow. (2021, December 19). *Convert MATLAB system identification to Python*. Retrieved from https://stackoverflow.com/questions/tagged/system-identification

- MathWorks. (n.d.). *System identification - File exchange - MATLAB Central*. Retrieved from https://www.mathworks.com/matlabcentral/fileexchange/170741-systemidentification