

**INDIAN INSTITUTE OF TECHNOLOGY KANPUR**  
**Department of Electrical Engineering**



**EE 496**  
**UNDERGRADUATE PROJECT I**

**Project Report**

**Project Title:**

**Simulating Multi-Microphone Spatial Audio in  
Reverberant Rooms**

**Supervisor:**

**Prof. Rajesh M. Hegde**

**Team Members:**

**Ayush Dixit (220262)**

**Ashwani Kumar Singh(220244)**

# Table of Contents

| S. No. | Section Title                         | Page No. |
|--------|---------------------------------------|----------|
| 1.     | Introduction                          | 3        |
| 2.     | Objective                             | 3        |
| 3.     | Tools and Libraries Used              | 4        |
| 4.     | Specific Array Toolbox Functions Used | 4        |
| 5.     | Room and Microphone Configuration     | 5        |
| 6.     | Audio Sources                         | 5-6      |
| 7.     | Room Impulse Response Simulation      | 6        |
| 8.     | Spatial Audio Generation              | 7        |
| 9.     | Gain Compensation                     | 7        |
| 10.    | Signal Alignment and Padding          | 7        |
| 11.    | Output and File Management            | 7        |
| 12.    | Outcomes                              | 7        |
| 13.    | Simulation Results                    | 8        |
| 14.    | Work Timeline                         | 9        |
| 15.    | Applications and Future Work          | 10       |
| 16.    | Conclusion                            | 10       |
|        | Important links                       | 11       |
|        | Appendix: Key Code Snippet            | 12-15    |
|        | Work Distribution                     | 16-17    |

## 1. Introduction

This project focuses on simulating realistic multi-microphone spatial audio recordings in a virtual room environment using MATLAB's Array Toolbox. The simulation involves audio sources representing a wearer and a bystander, recorded via a microphone array setup placed within a virtual room. Figure 1 represents the block diagram showing the flow of spatial audio generation. The key goal is to generate multi-channel audio that includes effects of room reverberation, source positioning, and ambient noise, suitable for applications in beamforming, speech enhancement, and audio source localization.

## 2. Objective

The primary objectives of this project are:

- To simulate a room with wearable and external audio sources
- To compute impulse responses using the image-source method
- To process real audio through these impulse responses
- To mix multiple convolved audio streams and add environmental noise
- To save the resulting multi-channel audio data for further analysis

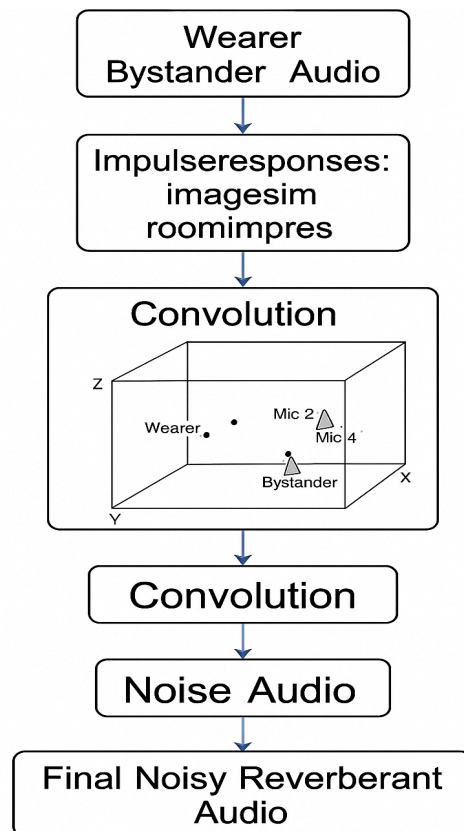
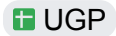


Fig:1 Block diagram representing the flow of spatial audio generation

### 3. Tools and Libraries Used

- **MATLAB Array Toolbox:** For simulating room acoustics using functions like `imagesim` and `roomimpres`
- **MATLAB Audio Toolbox:** For audio file handling and processing

Organized the usage and input - output of the functions into sheet for easier access



- **Custom MATLAB Scripts:** For audio normalization, convolution, mixing, and file management
- **LibriSpeech** to take clean audios and use them to generate convolved audios.
- **Python (pyroomacoustics):** Used initially for simulation in Python.
- **Custom Scripts:** For signal convolution, gain control, file output, and noise mixing.

### 4. Specific Array Toolbox Functions Used

The MATLAB Array Toolbox provided essential utilities for simulating the acoustic properties of a room. The following are the core functions used in this project:

- **`imagesim`:** This function implements the image source method to compute the delay times and amplitude scaling factors of sound reflections. It uses information about the room dimensions, source positions, microphone positions, and reflection coefficients to simulate how sound propagates in a reflective environment.
  - **Inputs:** Room vertices (`forv`), source and microphone coordinates, beta coefficients (reflection loss), speed of sound, and decibel cutoff.
  - **Outputs:** Delay times and scaling coefficients for each sound path.
- **`roomimpres`:** This function generates time-domain impulse responses from the delays and scaling coefficients calculated by `imagesim`. These impulse responses emulate the acoustic behavior of the room.
  - **Inputs:** Delay times and scaling factors, speed of sound, and sampling rate.
  - **Outputs:** A complete impulse response vector representing the direct and reflected paths of sound from source to microphone.

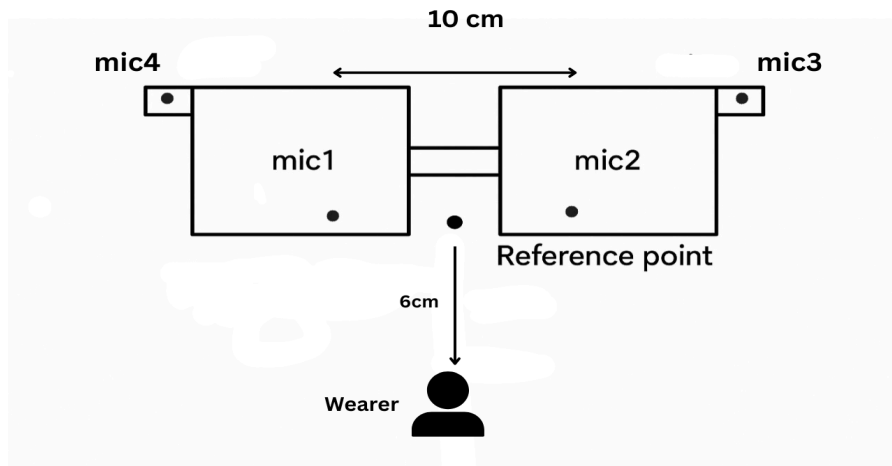
These functions enabled a precise simulation of room acoustics and microphone capture behavior, allowing for the realistic transformation of clean audio into reverberant multi-channel signals.

## 5. Room and Microphone Configuration

The room dimensions are set as 5m (length) x 4m (width) x 6m (height), centered at the origin to simplify coordinate calculations. The microphone array consists of four microphones placed as follows:

- Mic 1: At the center (0, 0, 0)
- Mic 2: 5 cm to the right of Mic 1 (0.05, 0, 0)
- Mic 3: Positioned at (0.08, 0.45, 0.04)
- Mic 4: Positioned symmetrically at (-0.08, 0.45, 0.04)

The wearer (audio source) is slightly offset from Mic 1 to simulate real-world wearable conditions.

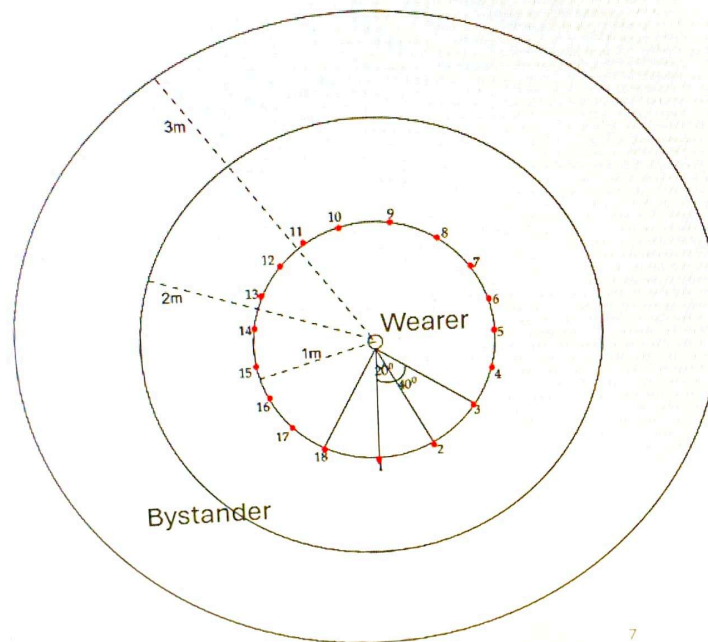


## 6. Audio Sources

Two speech recordings were used:

- Inputs:
  - Wearer audio: primary speaker
  - Bystander audio: secondary speaker
  - Noise samples: from FSDnoisy18k dataset
- Steps:
  - Normalize and resample all signals to 16 kHz.
  - Convolve wearer/bystander audio with respective IRs for each mic.
  - Gain compensation applied to match original loudness.
  - Signals padded to uniform length for mixing.
  - Final output: 4-channel noisy, reverberant audio + MAT matrix.

- **Wearer audio:** Representing the primary speaker
- **Bystander audio:** Positioned at varying distances and angles from the microphone array



## 7. Room Impulse Response Simulation

Using the Array Toolbox, the following steps were used to model acoustic propagation:

### 7.1 Image-Source Modeling with **imagesim**

This function computes delay times and scaling coefficients for direct and reflected paths between the source and microphones. Inputs include:

- Room corners (**forv**)
- Source and microphone positions
- Reflection coefficients (**bs**)
- Speed of sound ( $c = 343$  m/s)
- Decibel cutoff (**dbd** = 60 dB)

### 7.2 Impulse Response Generation with **roomimpres**

The output from **imagesim** is used to generate actual time-domain impulse responses. These are then padded or truncated to a fixed length (640 samples) to ensure uniformity across all recordings.

## 8. Spatial Audio Generation

For each impulse response, the following steps are taken:

- Each microphone channel convolves the source audio with its respective impulse response using 1D convolution (**conv**).
- Wearer and bystander convolved signals are summed for each channel.
- Environmental noise is randomly selected from a noise dataset folder and added to the mixed signal with a fixed scaling factor (0.0002).

## 9. Gain Compensation

To avoid excessive attenuation due to convolution, a fixed gain factor is applied to scale the resulting audio. This ensures a more realistic loudness level.

## 10. Signal Alignment and Padding

All audio components (wearer, bystander, noise) are padded to the maximum length of the longest signal to allow accurate mixing. Zero-padding ensures consistent length and format.

## 11. Output and File Management

Each unique configuration (defined by source radius and azimuth angle) results in:

- 4 WAV files (one per mic)
- 1 MAT file containing the combined multi-channel audio matrix

Files are saved using descriptive names indicating configuration parameters (e.g., radius and azimuth angle). E.g. **Audio10\_output\_r2\_phi60\_theta0\_mic3.wav**.

## 12. Outcomes

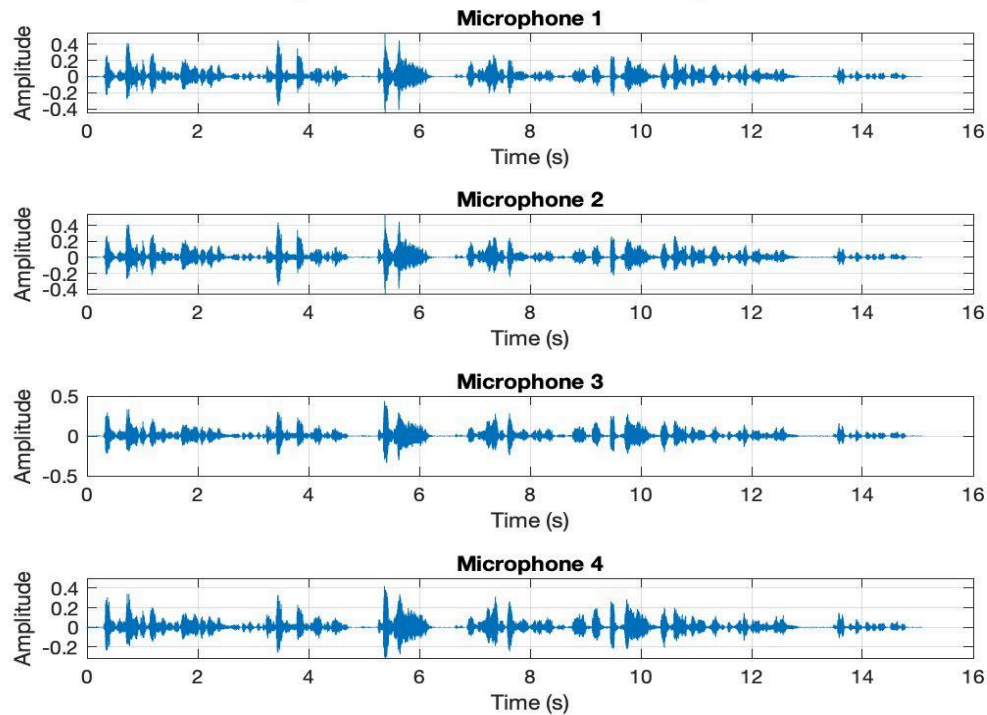
- Successfully simulated acoustic environments using MATLAB.
- Developed MATLAB code for flexible source/mic placement and convolution.
- Generated a large, labeled dataset of reverberant, multi-channel audio for both python and matlab.  
(<https://docs.google.com/spreadsheets/d/1zKVOFhMgzVvH5nOMFsRdMvYvvKgufhiGCdtnzWVZQ9I/edit?gid=0#gid=0> )

## • 13. Simulation Results

We created amplitude plots to compare the output audio signals received at the four microphone positions. The plots appeared largely similar in shape, with the primary differences being in amplitude levels—an aspect we have already discussed in detail.

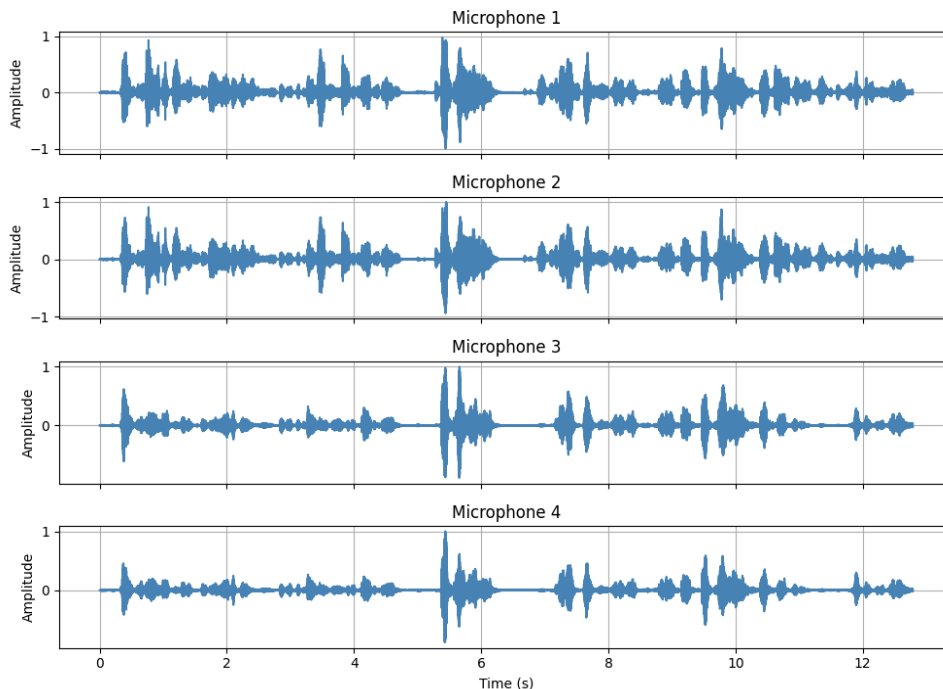
### 13.1 MATLAB

Amplitude vs Time for Audio 10,  $r=2$ ,  $\phi=100^\circ$



### 13.2 Python

Amplitude vs Time for  $r=2$ ,  $\phi=100^\circ$





## 14. Work Timeline

|                    |   |
|--------------------|---|
| <b>Weeks 1-2</b>   | <ul style="list-style-type: none"><li>• Literature review on beamforming and source localisation.</li><li>• Explored MATLAB's Array Toolbox and configured a 4-microphone array.</li><li>• Learn about different functions in the tool box and their correct usage.</li></ul>   |
| <b>Weeks 3-4</b>   | <ul style="list-style-type: none"><li>• Simulated RIRs using <code>imagesim</code> and <code>roomimpres</code> in MATLAB.</li><li>• Configured room and mic array geometries based on the room dimensions and provided information.</li><li>• Started working on Python code for RIR generation using <code>pyroomacoustics</code>.</li></ul>   |
| <b>Weeks 5-6</b>   | <ul style="list-style-type: none"><li>• Convolved RIRs with wearer and bystander speech signals.</li><li>• Mixed in environmental noise and visualized output with amplitude vs. time plots.</li><li>• Ran extensive simulations: 19 azimuth angles <math>\times</math> 3 radial distances.</li><li>• Produced ~10 hours of audio data and corresponding matrix files.</li></ul>  |
| <b>Weeks 7-8</b>   | <ul style="list-style-type: none"><li>• Encountered coordinate handling bugs (negative values not supported) for <code>pyroomacoustics</code>, used coordinate shifting to rectify the issue.</li><li>• Modified the coordinate system to shift the origin to centre of the room.</li></ul>   |
| <b>Weeks 9-10</b>  | <ul style="list-style-type: none"><li>• Added gain factor to increase the loudness.</li><li>• Ran simulations for 10 different clean audio combinations for each <math>r</math> and <math>\phi</math> for both.</li><li>• Produced ~10 hours of audio data and corresponding matrix files for both Python and MATLAB code and compared them on various parameters.</li></ul>  |
| <b>Weeks 11-12</b> | <ul style="list-style-type: none"><li>• Ran extensive simulations: 19 azimuth angles <math>\times</math> 3 radial distances for 50 different clean audio combinations for each <math>r</math> and <math>\phi</math>.</li><li>• Produced ~50 hours of audio data and corresponding matrix files</li><li>• Organized dataset and validated outputs.</li><li>• Documented process and prepared final report and results.</li></ul> |

## 15. Applications and Future Work

This simulated dataset can be used for:

- Training neural networks for speech enhancement or separation
- Testing microphone array algorithms (beamforming, source localization)
- Robustness testing in wearable audio capture research

Future improvements could include:

- Simulating directional microphones
- Incorporating variable reverberation times
- Modeling more complex room geometries
- Use the newly provided CPP code to simulate the same.
- Expand simulations to real-time and real-world scenarios.

## 16. Conclusion

This project successfully simulated a realistic multi-channel audio dataset by modeling room acoustics using both MATLAB and Python. The MATLAB Array Toolbox played a crucial role in generating precise Room Impulse Responses (RIRs) using `imagesim` and `roomimpres`, allowing us to model sound reflections and reverberations accurately. These RIRs were then convolved with clean audio to simulate the captured signals at each microphone, forming the core of the synthetic dataset.





In parallel, a Python-based approach using libraries such as `pyroomacoustics` offered similar capabilities, with greater flexibility for scripting and integration into larger machine learning pipelines. However, the MATLAB environment provided more stability and built-in tools for visualization and controlled simulation, especially when dealing with array geometries and acoustic path modeling.

When comparing results:

- **MATLAB** produced more interpretable and structured output due to its toolbox precision and GUI support, which was helpful during development and debugging.
- **Python**, while requiring more manual setup and tuning, allowed for more scalable dataset generation and integration with data handling and training pipelines (e.g., NumPy, pandas, PyTorch).

Both approaches successfully produced realistic multi-channel data for source localization and beamforming simulations, but each has its strengths—MATLAB excels in simulation accuracy and prototyping, whereas Python is better suited for automation, scalability, and ML workflow integration.

## Important Links:

- **Report-**  UGP
- **MATLAB and Python code -**  UGP Code
- **Input Audio Files -**  beamforming
- **Generated Data -**  
[https://iitk-my.sharepoint.com/:f:/g/personal/jsuraj\\_iitk\\_ac\\_in/EuD50DRfx2ZDvxOI8tMKnbwBN21QUvkg4Z41uwbunL2YPQ?e=NxUY63](https://iitk-my.sharepoint.com/:f:/g/personal/jsuraj_iitk_ac_in/EuD50DRfx2ZDvxOI8tMKnbwBN21QUvkg4Z41uwbunL2YPQ?e=NxUY63)
- **Array tool box functions -**  UGP

## Appendix: Key Code Snippet

### Link for Python Code:

[https://docs.google.com/document/d/1wTK2ZzMb\\_zRqbhEIA3NixAZSNK0AzcGN1yUQNSFOzC0/edit?tab=t.0](https://docs.google.com/document/d/1wTK2ZzMb_zRqbhEIA3NixAZSNK0AzcGN1yUQNSFOzC0/edit?tab=t.0)

### MATLAB Code:

```
clc;
clearvars;
%% Modified Room and Microphone Configuration
mic1pos = [0.05; 0; 0]; % Positioned relative to center
mic2pos = [-0.05; 0; 0]; % 10 cm right of mic1
mic3pos = [0.08; 0.45; 0.04]; % Positioned relative to center
mic4pos = [-0.08; 0.45; 0.04]; % Positioned relative to center
mics = [mic1pos, mic2pos, mic3pos, mic4pos];
%% Updated Room dimensions (centered at origin)
room_size = [5, 4, 6]; % Maintain original room dimensions
forv = [-room_size/2; room_size/2]'; % Symmetric around origin
% Now spans from [-5,-5,-1.5] to [5,5,1.5]
%% Modified Wearer Position (slightly offset from center)
sigpos_wearer = [0; -0.06; 0]; % Maintain original offset from mic1
%% Propagation properties
c = 343; % Speed of sound in m/s
dbd = 60; % Decibels down for cutoff
fs = 16000; % Sampling frequency in Hz
%% Define source positions
radii = [1, 2, 3]; % Radii for bystander movement
phi_angles = 0:20:360; % Azimuth angles for bystander movement
theta = 0; % Constant elevation angle (horizontal plane)
sigpos_wearer = [0; -0.06; 0]; % Wearer as speaker
L_target = 640; % Desired impulse response length
%% Load Wearer and Bystander Audio
[wearer_audio, fs_wearer] = audioread('1040-133433-0025.flac');
[bystander_audio, fs_bystander] = audioread('1040-133433-0009.flac');
% Resample audio if needed
if fs_wearer ~= fs
    wearer_audio = resample(wearer_audio, fs, fs_wearer);
end
if fs_bystander ~= fs
    bystander_audio = resample(bystander_audio, fs, fs_bystander);
end
%% Load Noise Files from Folder
noise_folder = '/Users/ASHWANI/Downloads/FSDnoisy18k.audio_test'; % UPDATE THIS PATH
```

```

noise_files = dir(fullfile(noise_folder, '*.wav')); % Add other extensions if needed
% Check if noise files exist
if isempty(noise_files)
    error('No noise files found in the specified folder.');
```

end

```

%% Create output folder
output_folder = 'final_noisy_outputs';
if ~isfolder(output_folder)
    mkdir(output_folder);
end
%% Set constant beta value
beta = 0.5;
bs = repmat(beta, 1, 6);
%% Main processing loop
for rIdx = 1:length(radII)
    r = radII(rIdx);
    for phi = phi_angles
        % Randomly select a noise file
        selected_noise = noise_files(randi(length(noise_files))).name;
        [noise_audio, fs_noise] = audioread(fullfile(noise_folder, selected_noise), fs);

        [x, y, z] = sph2cart(deg2rad(phi), deg2rad(theta), r);
        sigpos_bystander = [x; y; z];

        % Generate wearer IR
        IR_wearer = zeros(4, L_target);
        for micIdx = 1:4
            currentMic = mics(:, micIdx);
            [dIays, scls] = imagesim(forv, sigpos_wearer, currentMic, bs, c, dbd);
            [impres, ~] = roomimpres(dIays, scls, c, fs);
            impLen = size(impres,2);
            if impLen >= L_target
                impres_mod = impres(1, 1:L_target);
            else
                impres_mod = zeros(1, L_target);
                impres_mod(1:impLen) = impres(1, 1:impLen);
            end
            IR_wearer(micIdx, :) = impres_mod;
        end

        % Generate bystander IR
        IR_bystander = zeros(4, L_target);
        for micIdx = 1:4
            currentMic = mics(:, micIdx);
            [dIays, scls] = imagesim(forv, sigpos_bystander, currentMic, bs, c, dbd);
            [impres, ~] = roomimpres(dIays, scls, c, fs);
```

```

    impLen = length(impres);
    if impLen >= L_target
        impres_mod = impres(1:L_target);
    else
        impres_mod = [impres(:); zeros(L_target - impLen, 1)];
    end
    IR_bystander(micIdx, :) = impres_mod';
end

% Normalize IRs before convolution
ir_bystander = IR_bystander(micIdx, :) / max(abs(IR_bystander(micIdx, :)));
ir_wearer = IR_wearer(micIdx, :) / max(abs(IR_wearer(micIdx, :)));
final_noisy_audio = zeros(4, max([length(wearer_audio), length(bystander_audio),
length(noise_audio)]));

% Initialize convolution outputs
convolved_audio_bystander = zeros(4, length(bystander_audio));
convolved_audio_wearer = zeros(4, length(wearer_audio));
for micIdx = 1:4
    % Get correct impulse response for current microphone
    ir_bystander = IR_bystander(micIdx, :);
    ir_wearer = IR_wearer(micIdx, :);

    % Perform 1D convolution
    convolved_audio_bystander(micIdx,:) = conv(bystander_audio, ir_bystander, 'same');
    convolved_audio_wearer(micIdx,:) = conv(wearer_audio, ir_wearer, 'same');
end
% % Calculate energy compensation factor
original_rms = rms(wearer_audio);
convolved_rms = rms(convolved_audio_wearer);
%
%
% % Element-wise division with broadcast scalar
% % gain_factor = original_rms ./ (convolved_rms + eps);
gain_factor = 10;
%
% % Apply gain to each microphone channel
convolved_audio_wearer = convolved_audio_wearer .* gain_factor;
convolved_audio_bystander = convolved_audio_bystander .* gain_factor;
%
% Ensure matching lengths
max_length = max([size(convolved_audio_bystander,2),
    size(convolved_audio_wearer,2),
    length(noise_audio)]);

% Pad all signals to same length

```

```

convolved_audio_bystander = padarray(convolved_audio_bystander, [0
max_length-size(convolved_audio_bystander,2)], 0, 'post');
convolved_audio_wearer = padarray(convolved_audio_wearer, [0
max_length-size(convolved_audio_wearer,2)], 0, 'post');
noise_audio_padded = padarray(noise_audio(:)', [0 max_length-length(noise_audio)], 0, 'post');
% Add noise to final mixed audio
    noise_level = 0.0002;
    %   final_noisy_audio(micIdx, :) = final_mixed_audio + noise_level * noise_audio_padded;
    % end
% Create final mix
final_mixed_audio = convolved_audio_bystander + convolved_audio_wearer;
final_noisy_audio = final_mixed_audio + noise_level * noise_audio_padded;

    for micIdx = 1:4
        audiowrite(sprintf('%s/Audio10_output_r%d_phi%d_theta0_mic%d.wav', output_folder, r, phi,
micIdx), final_noisy_audio(micIdx, :), fs);
    end
    % Save the final noisy audio matrix
    save(sprintf('%s/Audio10_matrix_r%d_phi%d_theta0.mat', output_folder, r, phi),
'final_noisy_audio');
    fprintf('Processed r=%d, phi=%d° with noise: %s\n', r, phi, selected_noise);
end
end
disp('All combinations processed. Final noisy audio files and matrices saved successfully.');
```

```

function [y, fs] = audioresample(filepath, target_fs)
    [y, fs] = audioread(filepath);
    if fs ~= target_fs
        y = resample(y, target_fs, fs);
    end
end
end

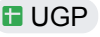
```

## WORK DISTRIBUTION:

|                                 |   |
|---------------------------------|---|
| <p>AYUSH DIXIT<br/>(220262)</p> | <ul style="list-style-type: none"><li>• Organized the usage, input &amp; output of the functions into sheets for easier access from the <b>Array tool box</b>.</li><li>• Convolved the <b>Room Impulse Responses (RIR)</b> for both wearer and bystander positions and implemented the entire signal processing pipeline in MATLAB code using <b>imagesim</b> and <b>roomimpres</b>.</li><li>• Computed the <b>varying coordinates</b> of the wearer and bystander based on the simulation design and organized all combinations into an Excel sheet. <b>beamforming</b></li><li>• Generated the output audio dataset with <b>10 audio combinations</b> corresponding to each position of bystander [radius (r) and azimuth angle (<math>\phi</math>)] and wearer.</li><li>• Python Work: Handled the Python coding part, from RIR generation to <b>convolution</b> and output generation.</li><li>• Implemented room simulation and RIR generation using the pyroomacoustics library in Python, defining <b>room geometry, absorption, and source/mic placement</b>.</li><li>• Addressed the limitation of pyroomacoustics in handling negative coordinates by adjusting the coordinate system to finalize the assignment of source and microphone positions for output generation.</li><li>• Performed <b>1D convolution</b> of clean audio signals with RIRs, scaled and mixed the wearer and bystander signals, and saved multi-channel outputs in .wav and .npy formats.</li><li>• Generated <b>10 hours</b> of output audio data across all microphones and source positions for <b>10 different audio combinations</b> of the wearer and bystander using the developed Python code.</li><li>• Visualized microphone outputs by plotting <b>Amplitude versus Time</b> for all 4 channels for Python code and compared the results with the output from MATLAB Code.</li></ul> |
|---------------------------------|---|



ASHWANI KUMAR  
SINGH  
(220244)

- Organized the usage, input, and output of all functions systematically into sheets. This improved accessibility and streamlined integration with the **Array Toolbox**.  UGP
- Generated the **Room Impulse Responses (RIRs)** separately for the wearer and bystander. Utilized functions from the Array Toolbox to create realistic reverberation profiles.
- Simulated realistic multi-channel reverberant audio through **1D convolution**. Clean audio signals were convolved with the generated RIRs at each microphone position.
- Enhanced dataset robustness by adding noise to the convolved audio samples. **Randomly selected realistic noise signals** from the provided noise database.
- Created an initial dataset by generating 10 audio combinations for **each  $r$  and  $\phi$** . This helped in capturing a variety of realistic audio scenarios early on.
- Encountered low loudness issues after shifting the room's **origin to the center**. Analyzed the root cause and observed significant drop in output signal amplitude.
- Resolved the loudness problem by applying **gain adjustment and zero-padding techniques**. This ensured that the clarity and dynamic range of signals were preserved.
- Expanded the dataset by generating **50 audio combinations** for **each  $r$  and  $\phi$  value**. The final dataset reached nearly **50 hours** of reverberant, noisy audio samples.
- Visualized the **amplitude versus time plots** for all four microphone channels. MATLAB-generated outputs helped in observing waveform patterns effectively.
- Compared the MATLAB waveform results with Python-based outputs for validation. Ensured **consistency and verified correctness** between the two implementations