EENG-320 Control Systems
Term Project 2
Members: Christian Calvo (1297165), and Erik Schlosser (1277551)
Due Date: 12/1/24

Git Repo: https://github.com/Tubliy/Proj2-Control-Systems

Objective:

Designing a camera system that can recognize the location of a sound source—such as a classroom speaker—and adjust its movement to follow it is the aim of this research. The speaker's position will be tracked via echolocation, which is the detection of sound waves and the calculation of the source's position based on the return time.

Design Parameters:

Input - The sound source is the only input.

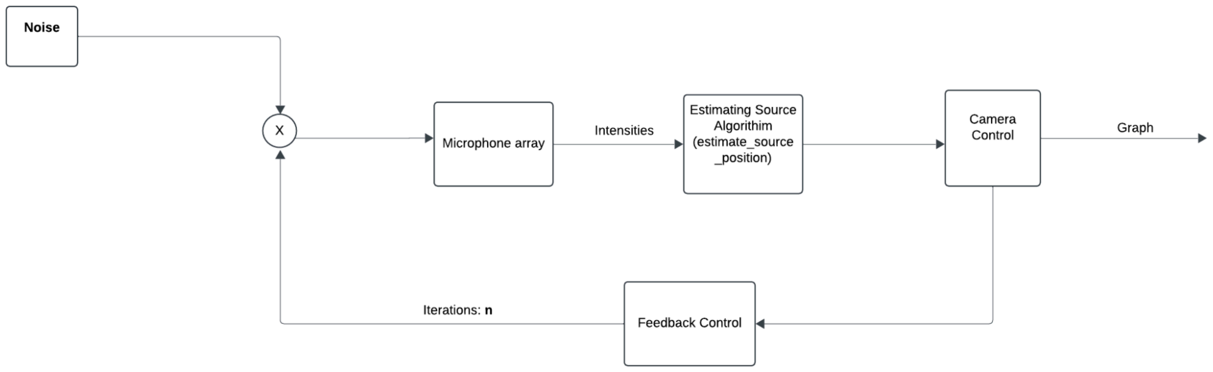
Output - Graph given after all conditions are met.

Microphone array - A total of four microphones set at each corner of the room.

Estimating Source Algorithm - Estimates the position of the source given the parameters microphone location, and the intensities.

Camera Control - Determines direction of camera using the estimated position and the camera's initial position of (0,0).

Feedback Control - Loop that iterates **n** times, and tracks the source position.



Procedure:

Set Up the Room:

- Define the dimensions of the room (width and height).
- Place four microphones at the corners of the room:
  - Top-left corner
  - Top-right corner

  - Bottom-left corner
  - Bottom-right corner

Initialize the True Source Position:

- Specify the initial coordinates (X, Y) of the sound source within the room.

Calculate Microphone Intensities:

- Use the inverse square law to compute the intensity of the sound signal at each microphone: $Intensity : \frac{1}{d^2}$
- Normalize the intensities so their sum equals 1.

Estimate the Source Position:

- Calculate the weighted centroid of the microphone positions using their intensities:

$$Estimated\ Position\ = \frac{\Sigma(Intensity \times Mic\ Position)}{\Sigma(Intensity)}$$

Visualize the System:

- Plot the following:
  - Microphone positions at the corners.
  - The true source position (orange).
  - The estimated source position (purple).
  - The camera position at the center (red) with an arrow pointing toward the estimated source.

Simulate Source Movement:

- Move the true source to a new position (e.g., increment its X-coordinate by 1 unit per iteration).

Iterate the Process:

- For a predefined number of iterations:
  - Recalculate microphone intensities.
  - Estimate the source position.
  - Visualize the updated positions and camera orientation.

Analyze Results:

- Compare the estimated source position with the true source position for each iteration.

- Observe the adjustments in the camera's orientation toward the estimated source.

Member Contributions:

Christian Calvo - Project2.py half, objective, procedure, and README file.

Erik Schlosser - Analysis, and Project2.py half

Works Cited:

"Spectrogram." *MathWorks*, www.mathworks.com/help/signal/ref/spectrogram.html. Accessed 30 Nov. 2024.

"Signal Processing Tutorial Launchpad." *MathWorks*, www.mathworks.com/academia/student_center/tutorials/signal-processing-tutorial-launchpad.html. Accessed 30 Nov. 2024.

"MATLAB Tutorial Launchpad." *MathWorks*, www.mathworks.com/academia/student_center/tutorials/mltutorial_launchpad.html?confirmation_page#. Accessed 30 Nov. 2024.

Analysis:

```
def estimate_source_position(mp, intensities):
    # Weighted centroid formula: Position = Σ(intensity * mic_position) / Σ(intensity)
    weighted_positions = np.array([intensity * np.array(mic) for
intensity, mic in zip(intensities, mp)])
    source_position = np.sum(weighted_positions, axis=0) /
np.sum(intensities)
    return source_position
```

Our first function measures the sound intensity in accordance with where the microphone is positioned and gives us an approximate location within the room for where the sound is coming from. We then convert the microphone positions into an array to be stored as a vector in order to complete future operations on these numbers.

```
      def calculate_microphone_intensities(mp, sp_estimate):
             # Use the inverse square law to estimate intensities
          distances = [np.linalg.norm(np.array(sp_estimate) -
np.array(mic)) for mic in mp]
          intensities = [1 / (d**2 + 1e-6) for d in distances]  # Avoid
division by zero
          normalized_intensities = intensities / np.sum(intensities)  #
Normalize
          return normalized_intensities
```

Our second function uses the estimated source position to then simulate the microphone intensity at that position. Mathematically speaking we are using the inverse square law here and setting our denominator to an incredibly small fraction to prevent any operations dividing by zero.

```
def visualize_camera_direction(room_size, mp, intensities, sp_estimate,
true_sp):

    width, height = room_size
    camera_position = (0, 0)  # Camera is at the center of the room

    # Calculate camera pointing direction
    dx, dy = sp_estimate[0] - camera_position[0], sp_estimate[1] -
camera_position[1]
    angle = np.degrees(np.arctan2(dy, dx))  # Angle in degrees for camera
orientation

    plt.figure(figsize=(8, 8))
    # Plot microphone positions
    for i, mic in enumerate(mp):
        plt.scatter(mic[0], mic[1], color='blue', label=f'Mic {i+1}' if i == 0 else None)
        plt.text(mic[0], mic[1] + 0.2, f'Mic {i+1} (Intensity: {intensities[i]:.2f})', fontsize=10, ha='center')

    # Plot estimated source position
    plt.scatter(sp_estimate[0], sp_estimate[1], color='purple', label='Estimated Source')
    plt.text(sp_estimate[0], sp_estimate[1] + 0.2, 'Estimated', fontsize=10, ha='center')

    # Plot true source position
    plt.scatter(true_sp[0], true_sp[1], color='orange', label='True Source')
    plt.text(true_sp[0], true_sp[1] + 0.2, 'True', fontsize=10, ha='center')

    # Plot camera position
    plt.scatter(camera_position[0], camera_position[1], color='red', label='Camera')
    plt.arrow(
        camera_position[0], camera_position[1],
        2 * np.cos(np.radians(angle)), 2 * np.sin(np.radians(angle)),  # Camera direction
        head_width=0.5, head_length=0.5, fc='green', ec='green', label='Camera Direction'
    )

    # Set plot limits and labels
    plt.xlim(-width / 2, width / 2)
    plt.ylim(-height / 2, height / 2)
    plt.axhline(0, color='black', linewidth=0.5, linestyle='--')
    plt.axvline(0, color='black', linewidth=0.5, linestyle='--')
    plt.title("Camera Direction Based on Estimated Sound Source and Microphone Inputs")
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.legend(loc="upper left")
    plt.grid(True)
    plt.show()
```

In the third function we are estimating the rooms layout in order for the microphones to have an assigned position within the output we see of the room and the sound positions. We plot the microphone positions on the graph and label them with corresponding intensity values. Once we see the true source positions of the noise we calculate an angle for the camera to point to and have an arrow facing in the direction of this vector angle. This function is also important in creating our legible plot with axes, legend, and grid to ensure a user friendly experience.

```
def feedback_loop(room_size, mp, true_sp, iterations=10, delay=1):

    sp_estimate = (0, 0)  # Start with an initial guess for the source
position

    for i in range(iterations):
        print(f"Iteration {i+1}:")

        # Simulate microphone intensities based on true source position
        intensities = calculate_microphone_intensities(mp, true_sp)

        # Estimate source position based on the intensities
        sp_estimate = estimate_source_position(mp, intensities)

        # Print estimated and true source positions
        print(f"Estimated Source Position: X = {sp_estimate[0]:.2f}, Y =
{sp_estimate[1]:.2f}")
        print(f"True Source Position: X = {true_sp[0]:.2f}, Y =
{true_sp[1]:.2f}")

        # Visualize the current state
        visualize_camera_direction(room_size, mp, intensities,
sp_estimate, true_sp)

        # Simulate movement of the true source (e.g., move right by 1
unit)
        true_sp = (true_sp[0] + 1, true_sp[1])  # Update true source
position
```

```
        time.sleep(delay)  # Wait for the next update
```

In the fourth function we are initializing our feedback loop into the system. Using the time import we want to simulate multiple iterations over a time period that reveals where the source position is estimated, visualized, and updated over these multiple iterations. To expand on this process, an initial estimate of the source position is made at the start of the process. The inverse-square law is then used to determine microphone intensities based on the actual source position. A weighted average of these intensities is then utilized to approximate the source position. A room layout is used to illustrate the current situation, including the estimated and actual positions. In order to simulate movement, the source position is then updated by increasing its x-coordinate. Finally, a pause is added before the subsequent cycle starts.

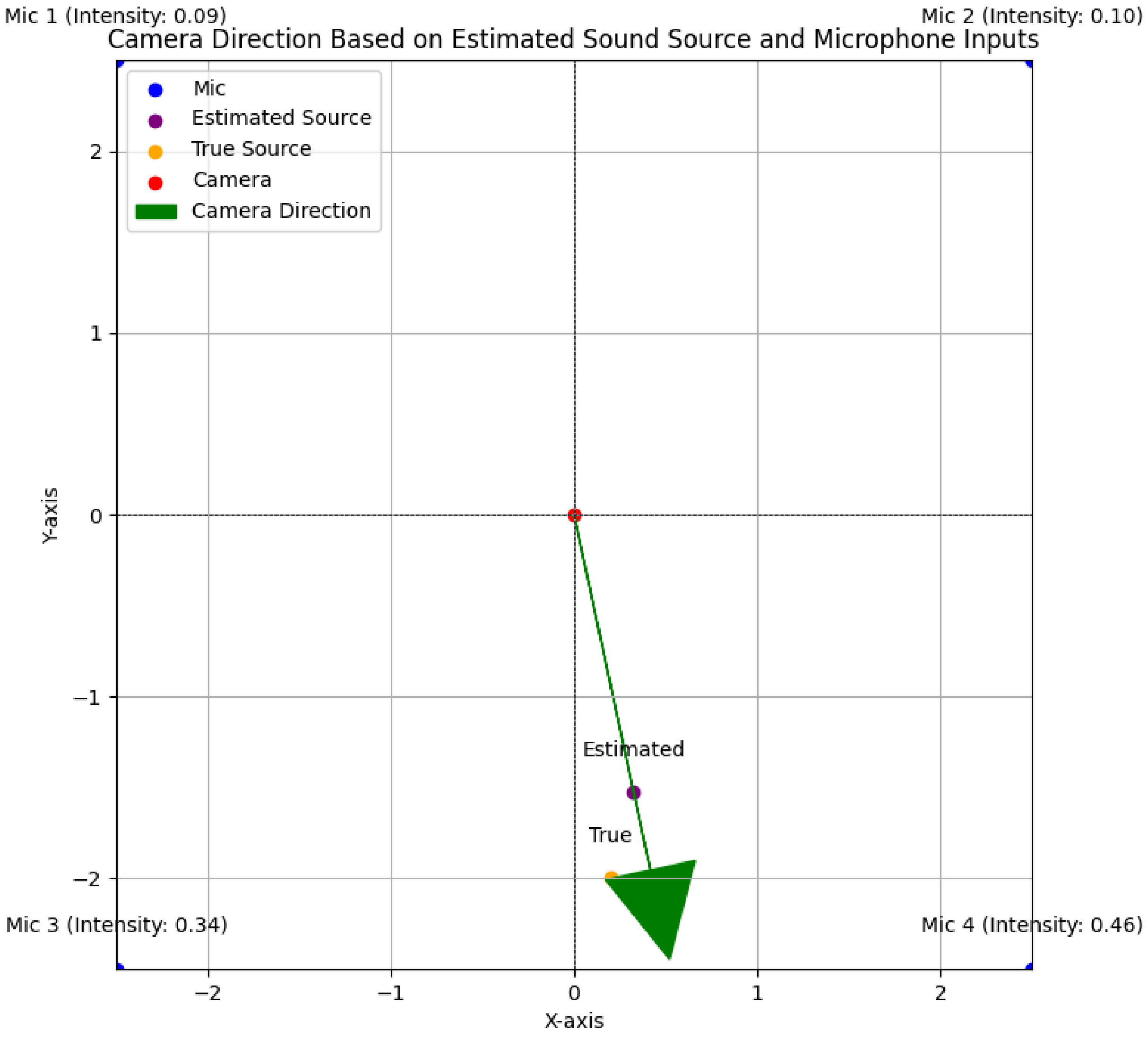
```
def main():
    # Room dimensions
    room_width = float(input("Enter the room width: "))
    room_height = float(input("Enter the room height: "))
    room_size = (room_width, room_height)

    # Initial true source position
    source_x = float(input("Enter the initial sound source X position: "))
    source_y = float(input("Enter the initial sound source Y position: "))
    true_sp = (source_x, source_y)

    # Microphone positions (corners of the room)
    mp = [
        (-room_width / 2, room_height / 2),  # Front-left
        (room_width / 2, room_height / 2),  # Front-right
        (-room_width / 2, -room_height / 2),  # Back-left
        (room_width / 2, -room_height / 2)  # Back-right
    ]

    # Simulate feedback loop
    iterations = int(input("Enter the number of iterations for the
feedback loop: "))
    delay = float(input("Enter the delay (in seconds) between iterations:
"))
    feedback_loop(room_size, mp, true_sp, iterations, delay)
```

The final function gathers input from the user to find the height and width parameters of the room along with a desired number of iterations with specified delays. It then uses the 4 corners of the room in order to set the microphone positions. Lastly it puts all this information into our feedback loop function and runs the program with all of the users desired inputs.



This is an example of what is given after the inputs, the microphone array is plotted in each corner. The true source is shown in orange and the estimated one is shown in purple. With each iteration the points are updated and we can see the estimated point getting closer to the true source. The camera is shown by the arrow with the starting origin of (0,0) and although it doesn't completely point to the true source, if we used a real camera it would be able to pick up the source. The microphone intensities are based on how close the true source is, the closer the higher intensity. This was found using the inverse square law, $Intensities = \frac{1}{d^2}$. The closer the distance the higher the intensity and vice versa.

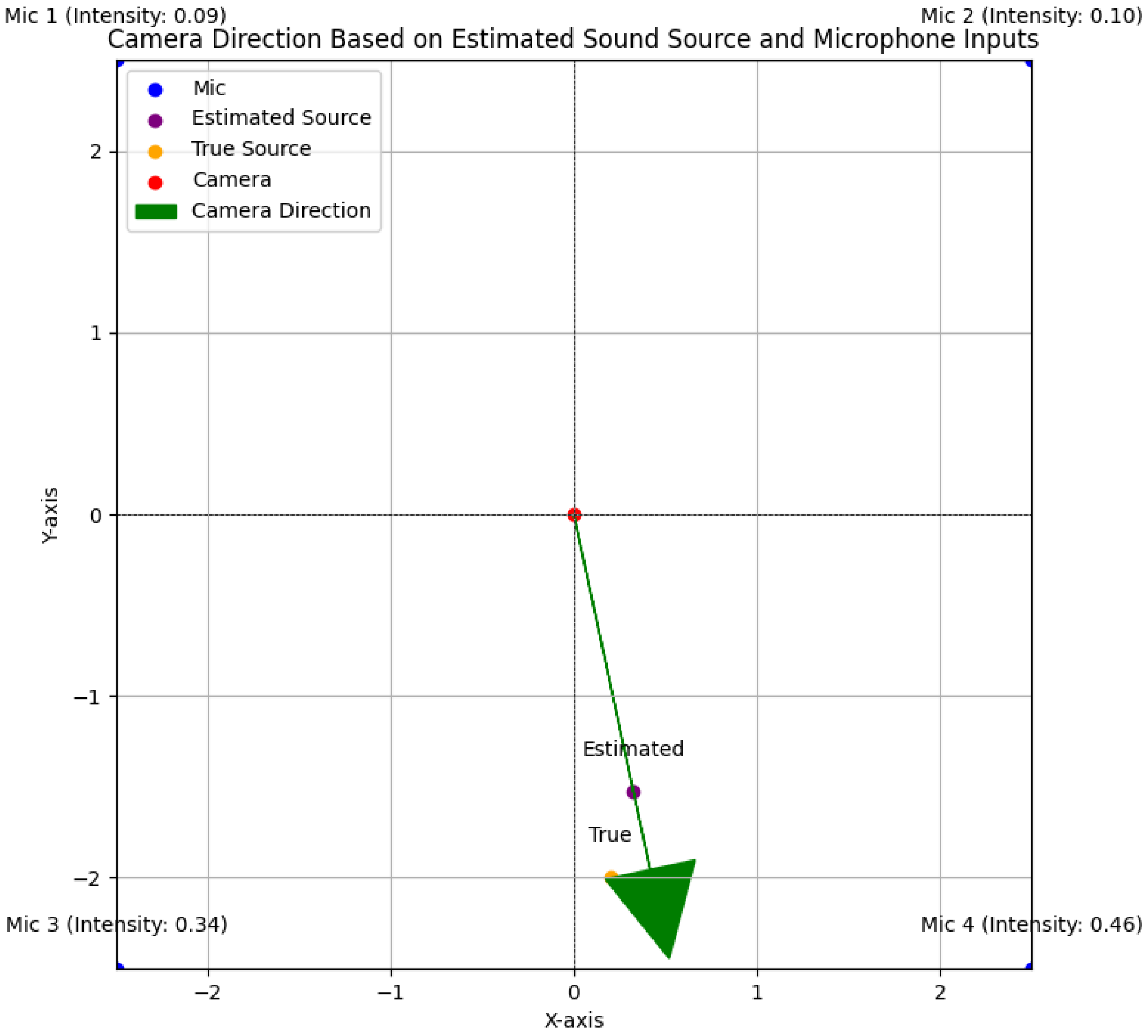
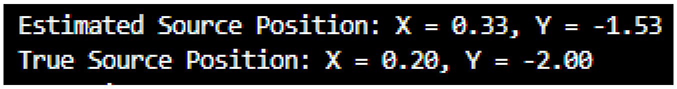


Figure 1A: First Iteration of Feedback Loop



```
Estimated Source Position: X = 0.33, Y = -1.53
True Source Position: X = 0.20, Y = -2.00
```

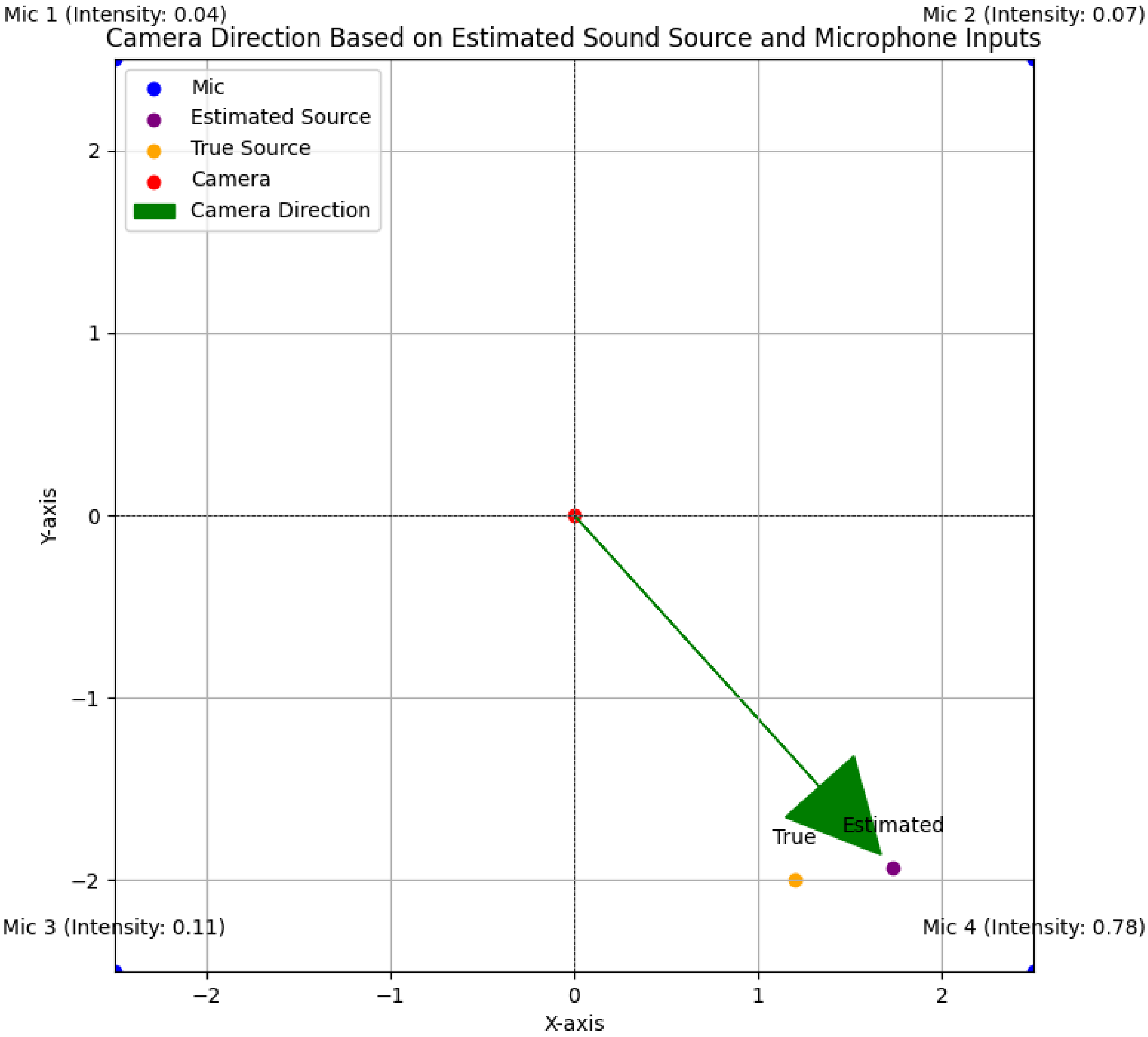
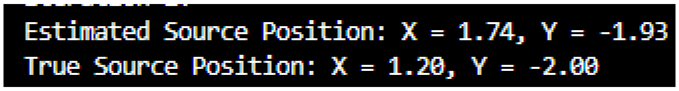


Figure 2A: Second Iteration of Feedback Loop



```
Estimated Source Position: X = 1.74, Y = -1.93
True Source Position: X = 1.20, Y = -2.00
```

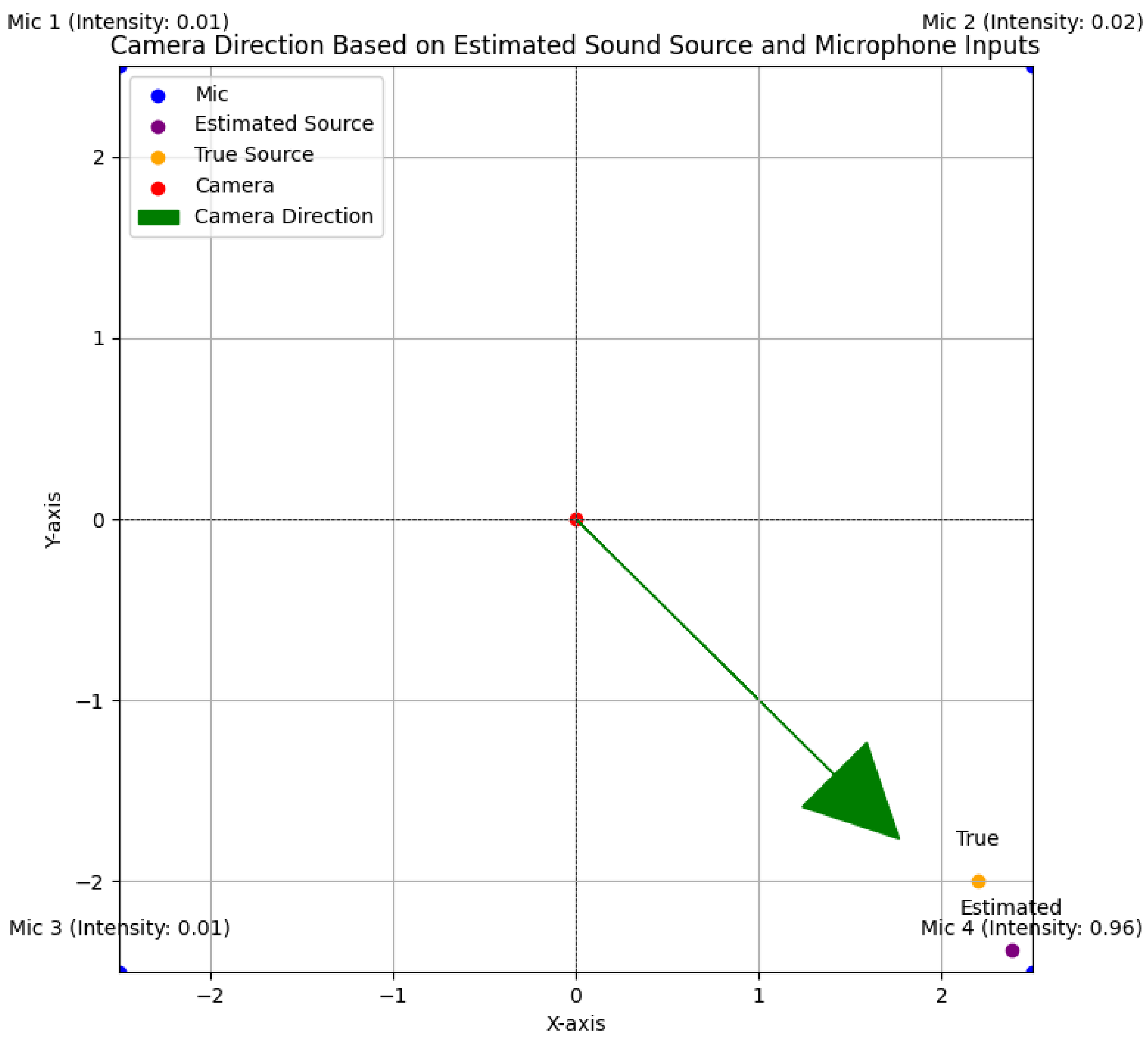
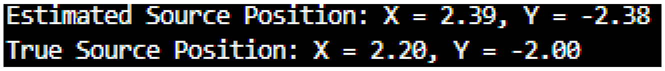


Figure 3A: Third Iteration of Feedback Loop



```
Estimated Source Position: X = 2.39, Y = -2.38
True Source Position: X = 2.20, Y = -2.00
```

This is after three iterations of the feedback loop, you can see how the estimated value is getting closer to the true source making the percentage error smaller. However, not being completely accurate it would still work in a real experiment.

Iterations **n**	True Source **X**	Estimated Source **X**	Percent Error
1	0.20	0.33	39.3939394%
2	1.20	1.74	31.0344828%
3	2.20	2.39	7.94979079%

Table for **X**, true and estimated source.

Iterations **n**	True Source Y	Estimated Source Y	Percent Error
1	-2.00	-1.53	23.5%
2	-2.00	-1.93	3.5%
3	-2.00	-2.38	19%

Table for **Y**, true and estimated source.

Appendix:

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


def estimate_source_position(mp, intensities):
    # Weighted centroid formula: Position = Σ(intensity * mic_position) / Σ(intensity)
    weighted_positions = np.array([intensity * np.array(mic) for intensity, mic in zip(intensities, mp)])
    source_position = np.sum(weighted_positions, axis=0) / np.sum(intensities)
```

```python
    return source_position


def calculate_microphone_intensities(mp, sp_estimate):
    # Use the inverse square law to estimate intensities
    distances = [np.linalg.norm(np.array(sp_estimate) - np.array(mic)) for mic in mp]
    intensities = [1 / (d**2 + 1e-6) for d in distances]  # Avoid division by zero
    normalized_intensities = intensities / np.sum(intensities)  # Normalize
    return normalized_intensities


def visualize_camera_direction(room_size, mp, intensities, sp_estimate, true_sp):

    width, height = room_size
    camera_position = (0, 0)  # Camera is at the center of the room

    # Calculate camera pointing direction
    dx, dy = sp_estimate[0] - camera_position[0], sp_estimate[1] - camera_position[1]
    angle = np.degrees(np.arctan2(dy, dx))  # Angle in degrees for camera orientation

    plt.figure(figsize=(8, 8))

    # Plot microphone positions
    for i, mic in enumerate(mp):
        plt.scatter(mic[0], mic[1], color='blue', label='Mic' if i == 0 else None)
        plt.text(mic[0], mic[1] + 0.2, f'Mic {i+1} (Intensity: {intensities[i]:.2f})', fontsize=10, ha='center')

    # Plot estimated source position
    plt.scatter(sp_estimate[0], sp_estimate[1], color='purple', label='Estimated Source')
    plt.text(sp_estimate[0], sp_estimate[1] + 0.2, 'Estimated', fontsize=10, ha='center')
    # Plot true source position
    plt.scatter(true_sp[0], true_sp[1], color='orange', label='True
Source')
    plt.text(true_sp[0], true_sp[1] + 0.2, 'True', fontsize=10,
ha='center')

    # Plot camera position
    plt.scatter(camera_position[0], camera_position[1], color='red',
label='Camera')
    plt.arrow(
        camera_position[0], camera_position[1],
        2 * np.cos(np.radians(angle)), 2 * np.sin(np.radians(angle)),  #
Camera direction
        head_width=0.5, head_length=0.5, fc='green', ec='green',
label='Camera Direction'
    )

    # Set plot limits and labels
    plt.xlim(-width / 2, width / 2)
    plt.ylim(-height / 2, height / 2)
    plt.axhline(0, color='black', linewidth=0.5, linestyle='--')
    plt.axvline(0, color='black', linewidth=0.5, linestyle='--')
    plt.title("Camera Direction Based on Estimated Sound Source and
Microphone Inputs")
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.legend(loc="upper left")
    plt.grid(True)
    plt.show()


def feedback_loop(room_size, mp, true_sp, iterations=10, delay=1):

    sp_estimate = (0, 0)  # Start with an initial guess for the source
position

    for i in range(iterations):
        print(f"Iteration {i+1}:")
        # Simulate microphone intensities based on true source position
        intensities = calculate_microphone_intensities(mp, true_sp)

        # Estimate source position based on the intensities
        sp_estimate = estimate_source_position(mp, intensities)

        # Print estimated and true source positions
        print(f"Estimated Source Position: X = {sp_estimate[0]:.2f}, Y =
{sp_estimate[1]:.2f}")
        print(f"True Source Position: X = {true_sp[0]:.2f}, Y =
{true_sp[1]:.2f}")

        # Visualize the current state
        visualize_camera_direction(room_size, mp, intensities,
sp_estimate, true_sp)

        # Simulate movement of the true source (e.g., move right by 1
unit)
        true_sp = (true_sp[0] + 1, true_sp[1])  # Update true source
position

        time.sleep(delay)  # Wait for the next update


def main():
    # Room dimensions
    room_width = float(input("Enter the room width: "))
    room_height = float(input("Enter the room height: "))
    room_size = (room_width, room_height)

    # Initial true source position
    source_x = float(input("Enter the initial sound source X position: "))
    source_y = float(input("Enter the initial sound source Y position: "))
    true_sp = (source_x, source_y)

    # Microphone positions (corners of the room)
    mp = [
        (-room_width / 2, room_height / 2),  # Front-left
        (room_width / 2, room_height / 2),  # Front-right
        (-room_width / 2, -room_height / 2),  # Back-left
        (room_width / 2, -room_height / 2)  # Back-right
    ]

    # Simulate feedback loop
    iterations = int(input("Enter the number of iterations for the
feedback loop: "))
    delay = float(input("Enter the delay (in seconds) between iterations:
"))
    feedback_loop(room_size, mp, true_sp, iterations, delay)


if __name__ == "__main__":
    main()
```