Acoustic Source Localization and Optimization Using Cardioid Microphone Sensors

Rosy Lin (RL792)

Aidan Singh (AAS386)

Robot Perception

December 8th, 2023

1 Abstract

This project builds on our prior work in sensor simulation, aimed at pinpointing sound source locations using a pair of coincident microphones. It showcases the enhanced capability of this sensor system in optimizing sound source localization. The simulation not only predicts and estimates the direction and distance of sound sources relative to the microphones but also integrates advanced features to effectively handle sound source angles and multiple sound sources. Key achievements include the accurate modeling of microphone gain patterns, precise estimation of source direction and distance, adaptive microphone orientation for optimal sound source angle detection, and strategic optimization for proximity-based sound source targeting. The success of this project underscores the efficiency and applicability of our sensor simulation methodology.

2 Introduction

This section outlines the methodology and theoretical underpinnings of our project, which extends our previous work in estimating sound source locations using a coincident microphone pair. Coincident microphone pairs, akin to binaural microphones, are instrumental in capturing spatial sound information, mirroring the way our auditory system processes sound. Building upon this foundational concept, our project's expanded goal is to leverage the distinctive attributes of coincident microphone pairs not only to accurately determine the direction and distance of sound sources but also to optimize sound source localization in a dynamic environment with multiple sources. This approach involves sophisticated modeling of microphone gain patterns and the implementation of adaptive strategies for microphone orientation, thereby enhancing the accuracy and efficiency of sound source estimation.

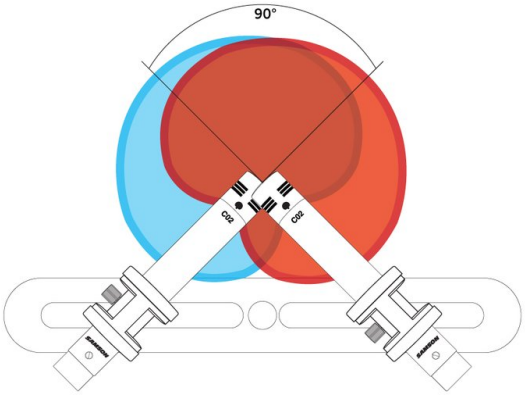


Figure 1: Coincident X-Y Microphone Pair

3 Theory and Methods

This section serves as the foundation for understanding the principle, mathematical model, and simulation techniques used in achieving the simulation objectives.

3.1 Background

In the coincident X-Y configuration, two directional cardioid microphones are placed so that their capsules are as close together as possible without touching. The principle behind the coincident X-Y technique is phase coherence. Since the capsules are so close together, any sound source will reach both mics at essentially the same time. This means that there are no time-based phase differences between the channels. In the case for this project, the microphones are oriented at an angle of 90 degree angle relative to each other, and we assume that the source comes from the front 180 degrees of the x-y system with a gain of 1 at the front and 0 at the back. The cardioid pattern uses "$gain = 0.5$" as the amplitude multiplier, which means that the output of the system has an amplitude half as large as the input.

3.2 Theory and Mathematical Models behind Optimization

3.2.1 Control Theory in Microphone Orientation

Feedback Control Systems: Previously, the microphones' sensitivity varies with the angle of the sound source. Now, we optimize the measurement by dynamically adjusting the angles of the microphones based on previous detections to achieve optimal localization of sound sources. The core of our method lies in feedback control, where the system's output (microphone pair orientation) is adjusted based on the input (detected sound source position). Given that the microphones are maintained at a 90-degree X-Y configuration, the optimal angle to the source is at a 45-degree angle relative to the microphone's axis.

3.2.2 Multi-Source Handling

Spacial Audio Processing: This involves distinguishing and processing sounds from different spatial locations. Beamforming techniques are employed to focus on specific sound sources while minimizing interference. The idea is to spatially separate distinct sound sources and process them individually. To optimize, we choose the closest sound source to orient the microphone pair towards by calculating the Euclidean distance of each sound source to the microphone pair.

3.3 Key Functions and Equations (see attached code for implementation):

- $nearest\_source$: Find the nearest source to the microphone pair
- $source\_angle$: Calculate the angle to the nearest source

- *optimize_sensor_angles*: Optimizes sensor by utilizing the above 2 variables - picking the closest source to each microphone and adjusting the angle of the microphone to place the source at a 45-degree angle from the microphone's axis
- *adjust_angle*: Adjusts the microphone pair's angle to the optimal angle calculated above

Part 1: Foundational equations for sound source prediction and estimation:

1. **Sound Source Angle Calculation:** this equation estimates the angle between the microphone pair's axis and the line connecting the source to the microphones.

$$optimal\_angle = \arctan 2(y_{nearest\_source} - y_{mic}, x_{nearest\_source} - x_{mic}) \tag{1}$$

2. **Microphone Gain Calculation:** the gain pattern of each microphone is modeled as a cardioid pattern, which reflects the microphone's sensitivity to sound arriving from different angles. It is higher when the sound source is directly in front of the microphone ($\theta = \theta_{\text{mic}}$) and decreases as the source moves away from the microphone's axis.

$$G(\theta) = \frac{1}{2}\left(1 + \cos(\theta - \theta_{\text{mic}})\right) \tag{2}$$

3. **Sound Source Distance Calculation:** this equation uses inverse square law (the received sound power is inversely proportional to the square of the distance) to calculate the Euclidean distance between the microphone and the sound source.

$$d = \sqrt{(x - xpos_{\text{mic}})^2 + (y - ypos_{\text{mic}})^2} \tag{3}$$

4. **Microphone Gain Adjustment for Distance:** To account for the effect of distance on the received sound power, the microphone gain can be adjusted using the inverse square law.

$$G_d(\theta) = \frac{1}{d^2} G(\theta) \tag{4}$$

where

- x and y: Cartesian coordinates of the sound source.
- θ: Angle (in radians) between the microphone pair's axis and the line connecting the source to the microphones.
- d: Distance between the sound source and the microphone pair.
- $G(\theta)$: Gain pattern of a single microphone in the pair as a function of θ.
- G_1 and G_2: Gains of the first and second microphones in response to the sound source.

Part 2: In addition to the above estimation models for the microphone simulation, we employ the below equations as additional features that optimize localization.

5. **Finding the Nearest Source:**

$$\text{nearest\_source} = \min_{s \in \text{sources}} \sqrt{(s_x - \text{mic}_x)^2 + (s_y - \text{mic}_y)^2} \tag{5}$$

where

- s_x and s_y represent the x and y coordinates of a source, respectively
- mic_x and mic_y are the x and y coordinates of the microphone
- The expression under the square root calculates the Euclidean distance between the microphone and a source
- min finds the source s for which this distance is minimized, identifying the nearest source

6. **Calculating the Angle to the Nearest Source:**

$$\text{source\_angle} = \arctan 2(\text{nearest\_source}_y - \text{mic}_y, \text{nearest\_source}_x - \text{mic}_x) \tag{6}$$

7. **Adjusting the Microphone Angle:**
For even-indexed microphones:

$$\text{optimal\_angle} = \text{source\_angle} - \frac{\pi}{4} \tag{7}$$

For odd-indexed microphones:

$$\text{optimal\_angle} = \text{source\_angle} + \frac{\pi}{4} \tag{8}$$

where

- optimal_angle is the new microphone angle
- source_angle is the current angle
- $\frac{\pi}{4}$ is the adjusted angle given the microphone pair's assumed 90-degree configuration

4 Procedure

This section outlines the step-by-step procedure for simulating and optimizing sound source location using a coincident microphone pair. The virtual sensor simulation was implemented using Python. We created instances of the microphone sensor, placed them at specific coordinates, and simulated a sound source at an arbitrary location.

The project encompasses the following steps:

1. **Microphone Class**: A Python class, 'mic,' was created to represent the coincident microphone pair. This class encapsulates properties such as microphone positions and angles, gain

patterns, and methods for estimating gains from sound sources as well as adjusting angle towards sound source for optimization.

2. **Sound Source Estimation**:
   Angle calculation: utilize the `np.arctan2` function to calculate the angle between the microphones' axis and the line connecting the sound source to the microphones. Equation (1)

   Gain calculation: calculate the gain for each microphone in response to the sound source's position. The gains can be obtained based on the angle (θ) and distance (d) from the microphones to the source. The code provided includes a cardioid pattern. Equation (2)

   Distance Estimation: estimate the distance (d) between the sound source and the microphone pair using the inverse square law and the calculated gains. For case when multiple sound sources are present, calculate the minimum distance for optimization. Equation (3), (4), and (5)

3. **Virtual Sensor Simulation:** A simulation environment was established to mimic the operation of a real coincident microphone pair. The simulation environment includes the placement of microphones and sound source coordinates. The simulation setup is then plotted as a graph for visualization.

4. **Sound Source Estimation and Optimization:** Simulate sound source localization by providing arbitrary sound source coordinates. Calculate the gains and estimate the sound source's angle and distance using the developed methods. Optimization is achieved by orientating the microphone pair towards the closets sound source using the additional features. Equation (5)-(8)

5 Results

In the development and assessment of the sensor system, several components were designed and validated. These components were crucial in helping us understand and evaluate the efficacy of our multi-variable controller.

5.1 Direction and Distance Analysis for Microphone Pair Using Gain Ratios

As a part of previous development, we determine the direction of a sound source by leveraging the gains from two microphones. Plots produced from this function showed two cardioid gain patterns with an overlaid dot representing the estimated source location. This visual representation provides an understanding of the source's position relative to the two microphones.

```
5.11
Source at (1, 1), outputs:
Estimated angle in radians: 0.7854
Estimated distance: 1.4142
```

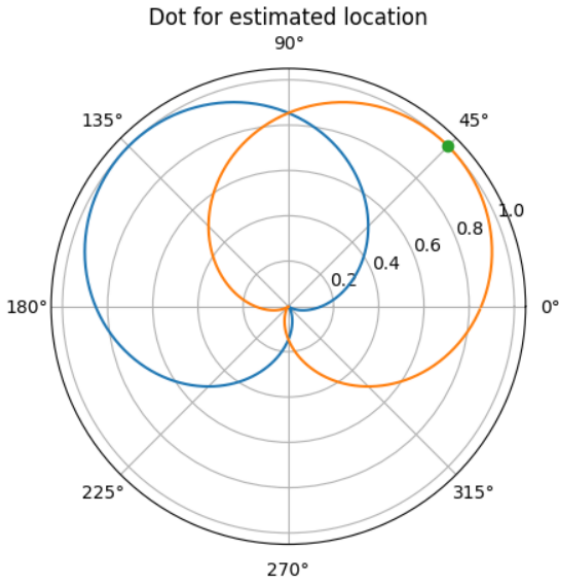


Figure 2: Estimated Source for getDirectionFromGains x = 1, y = 1

```
5.12
Source at (-1, 2), outputs:
Estimated angle in radians: 2.0420
Estimated distance: 2.2361
```

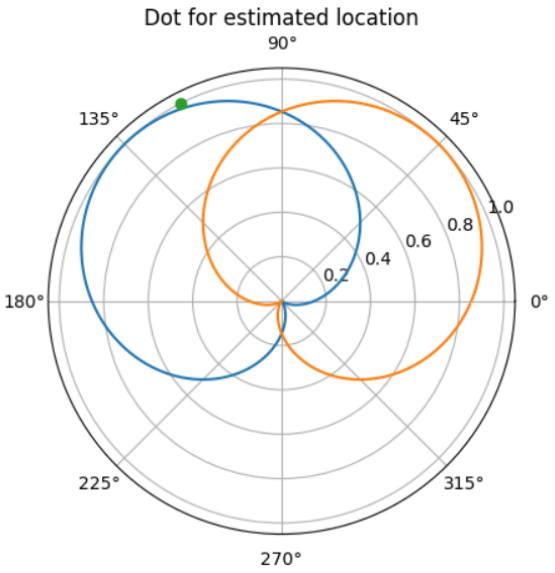
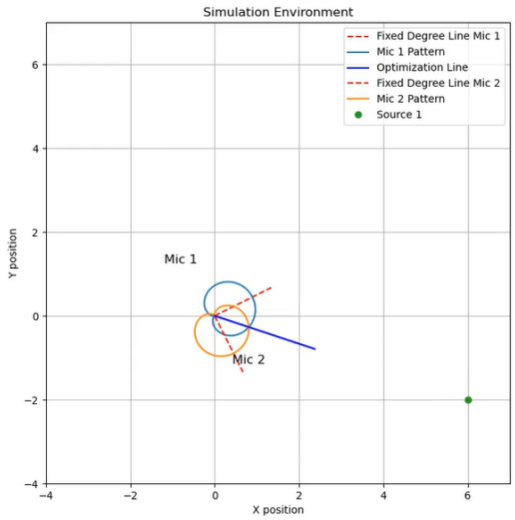


Figure 3: Estimated Source for getDirectionFromGains x = -1, y = 2

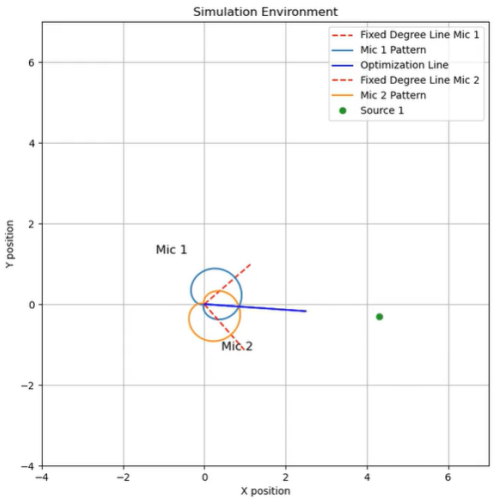
5.2 Analysis of Microphone Pair Orientation Optimization for a Single Sound Source

The core objective of the microphone pair orientation optimization is to ensure the most efficient capture of sound from a given source. In the context of a single sound source, the optimization algorithm adjusts the orientation of each microphone to maximize the directivity gain towards the source. The process is governed by the cardioid pickup pattern of the microphones, which is most sensitive at the front and least sensitive at the rear. Optimization is achieved by adjusting the orientation of each microphone to maintain a 45-degree angle relative to its nearest sound source. This approach ensured that each microphone was optimally positioned to capture sound from its proximate source while maintaining a consistent 90-degree X-Y configuration between the two microphones.

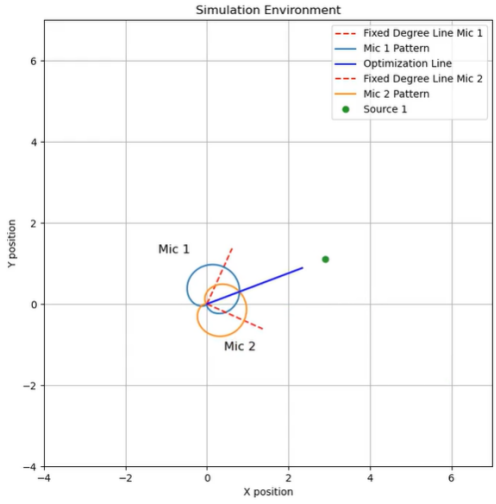
The optimization process involves calculating the angle from each microphone to the sound source and then adjusting the microphone's orientation to position the source optimally within its pickup pattern. Below are frames of our media visualization (video attached to project) of how microphone orientation can be adjusted in real-time for optimal sound source capture.



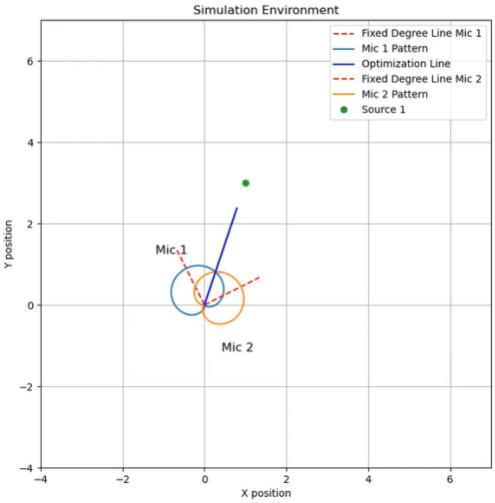
(a) Frame 1



(b) Frame 2



(a) Frame 3



(b) Frame 4

Figure 5: Video Frames of Mic Orientation in Real-time

5.3 Analysis of Microphone Pair Optimization for Multi-Source Handling

Expanding upon orientation optimization, the simulation extends beyond the capture of sound from a singular source to managing the interplay of multiple sources, optimizing the microphones' orientations for optimization. The optimization was achieved through a two-fold approach: First, by identifying the closest sound source to the microphone pair using Eucleadian distance and second, by adjusting the orientation of each microphone to maintain a 45-degree angle relative to its nearest sound source.

For visualization, we produced a initial configuration (before) plot and a post-optimization (after)

plot, which clearly demonstrated the effectiveness of the approach. Red dashed lines indicated the orientation of each microphone, while blue lines showed the direct 45-degree angle to the closest sound source. Additionally, cardioid patterns around each microphone visually represented their sensitivity and directional response characteristics.

```
5.31 For microphone pairs at origin (0,0)
    Initial angle for Mic 1: 90 degree
    Initial angle for Mic 2: 0 degree
```

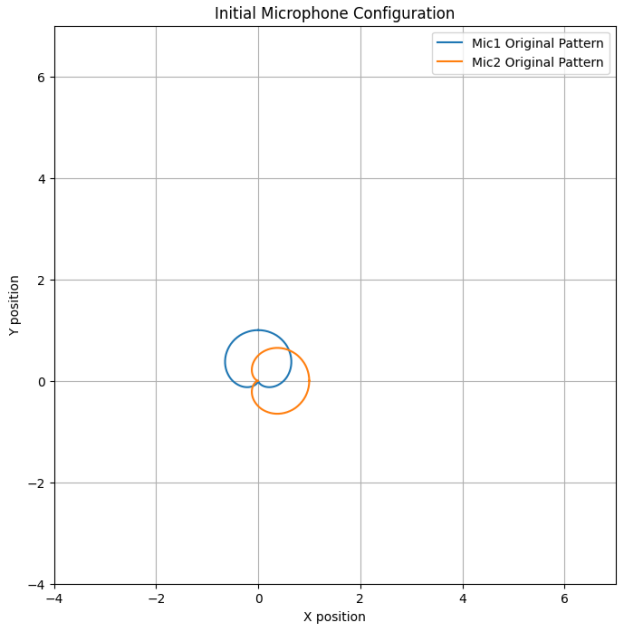


Figure 6: Initial Configuration for Microphone Pair

```
5.32 Source 1: (1.5, 4);    Source 2: (-2, 6)
    Closest source identified: source 1
    Source Angle Relative to x-axis: 69.44 degrees
    Rotation needed for Microphone Paire: 24.44 degrees
```

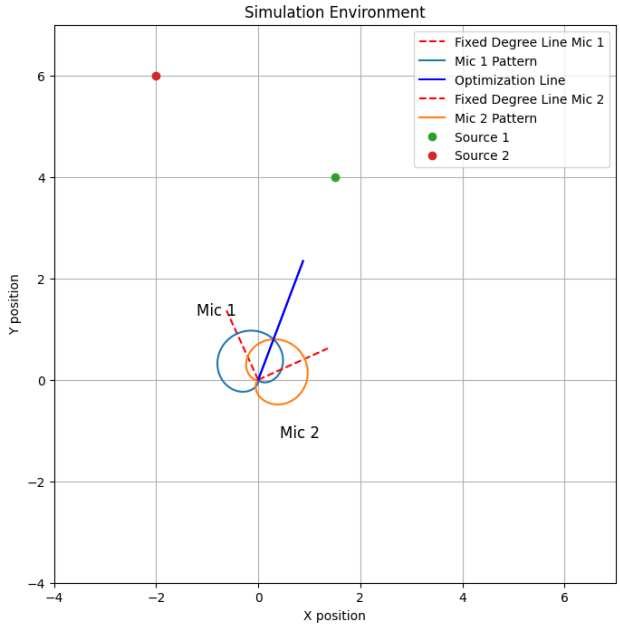


Figure 7: Optimization of Microphone Pair with Angle and Source Adjustments

5.4 Analysis on Sensor Properties and Dependencies

Per requirement for ECE 5240, the sensor model captures the sensor properties and the dependencies in Figure 1 of the Midterm Project description for the following reasons.

5.4.1 Sensor Model Selection

The sensor is a cardioid microphone with polar response:

$$G(\theta) = 0.5(1 + \cos(\theta - \theta_{\text{mic}}))$$

5.4.2 Sensor Dependencies

- Operating Conditions, A: Defined by ‘angle‘ and position (‘xpos‘, ‘ypos‘).
- Sensor State, X: ‘angle‘ captures microphone’s orientation.
- Sensor Measurements, Z: Determined in ‘getGainFromSource‘.

5.4.3 Target Dependencies

- **Target Features,** X': Angle of arrival (AoA) given by:

$$\theta_{\text{source}} = \arctan 2(y_{\text{source}} - y_{\text{mic}}, x_{\text{source}} - x_{\text{mic}})$$

Distance to the source:

$$d = \sqrt{(y_{\text{source}} - y_{\text{mic}})^2 + (x_{\text{source}} - x_{\text{mic}})^2}$$

- **Target State,** X_r: Estimated direction from 'getDirectionFromGains' and distance from 'getDistance'.

5.5 Expansion and Possible Causes of Error

5.5.1 Discretization Error Calculation:

In terms of error measurements, we calculate the discretization error on the gain values. Given an actual gain and a lookup table of possible gain values (which represent the discretized versions), it finds the closest value in the lookup table to the actual gain. Below is a table of the original gain and quantized gain values for the positions of the simulated microphone pair thus far:

x	y	original gain	quantized gain
1	0	0.5	0.499499499499499
1	1	0.426776695296637	0.426426426426426
3	3	0.047419632810737	0.047047047047047
-1	2	0.189100652418837	0.189189189189189

Figure 8: Table of Discretization Values for Given x and y

5.5.2 Inclusion of Environmental Noise:

Building upon our previous work, we added a *add_noise_to_signal* function that aims to add a layer of realism to the simulation by incorporating environmental noise into the otherwise clean audio signal. This is essential for testing and optimizing microphone configurations under more realistic conditions, where environmental noise is always a factor.

```
noise = np.random.normal(0, noise_level, len(signal))
```

The function uses *np.random.normal* to generate random noise. This function creates a series of random values that follow a normal (Gaussian) distribution.

5.5.3 Possible Causes for Errors:

While the system performed effectively in simulations, there are some improvements and potential sources of error that can significantly enhance the simulation's realism and applicability to real-world scenarios:

1. Dynamic Angle Optimization: While the current project allows for optimization, the model can be improved by implementing algorithms such as Neural Networks that can dynamically optimize the angle of the microphones based on real-time analysis of the acoustic environment.

2. Model Pattern Simplification: The randomized noise feature to simulate an environmental effect as well as the quantization error to simulate the discretization errors are both simplification of real-world patterns, which can be complex and non-uniform. The method for determining the closest source may not account for reflective surfaces or obstacles that could affect sound propagation.

3. Limitations of the Cardioid Model: The current system assumes a perfect cardioid response. However, real microphones may have imperfections as well as deviations from this pattern due to manufacturing variances or design differences.

6 Conclusion

In the culmination of our project, we successfully designed a coincident X-Y microphone-based system that leverages cardioid polar patterns to determine the angle and distance of sound sources. We further demonstrated the simulation's ability to optimize microphone angles relative to sound sources. Key achievements of this project include the development of methods for adding realistic environmental noise, calculating gains based on microphone orientation and source position, and optimizing microphone angles for improved sound pickup. The simulation environment provides a visual and analytical framework to understand how different factors affect the performance of a cardioid microphone pair.

As with any simulation, the project's current scope has limitations that offer avenues for future enhancement. The simplifications made in noise modeling, distance calculations, and cardioid pattern representation are necessary for a manageable simulation scope but do not fully encapsulate the complexities of real-world acoustics. Future work can focus on refining these aspects, incorporating more advanced noise models, dynamic angle optimization, and real-world data validation to enhance the simulation's accuracy and applicability. With continued refinement and enhancement, this tool has the potential to become an invaluable resource for audio professionals, researchers, and enthusiasts aiming to optimize microphone setups in various environments.

7 References

[1] "4 Stereo Microphone Recording Techniques," Samson Technologies Blog. [Online]. Available: https://samsontech.com/blog/4-stereo-microphone-recording-techniques/. [Accessed: Oct-16-2023].

8 Appendix

```
import numpy as np
import matplotlib.pyplot as plt
import os
os.chdir('video')
import subprocess
import glob

class mic:

    def __init__(self, name, xpos, ypos, angle):
        self.name = name
        self.xpos = xpos
        self.ypos = ypos
        self.angle = angle

    def add_noise_to_signal(self, signal, noise_level):
        # adds environmental noise to the signal.
        noise = np.random.normal(0, noise_level, len(signal))
        return signal + noise

    def getAngleGainVectors(self):
        radians = np.arange(0,2*np.pi,2*np.pi/360)
        return .5*(1 + np.cos(radians-self.angle)), radians


    def getGainFromSource(self, xpos, ypos, noise_level = 0, graph=True):
    # determines angle

        # get angle (radians) from source
        source_angle = np.arctan2(ypos-self.ypos,xpos-self.xpos) % (2*np.
    pi)

        # get difference in source and self angles
        angle_diff = self.angle-source_angle

        #calculate gain from cardioid polar pattern
        radians = np.arange(0,2*np.pi,2*np.pi/360)
        gains = .5*(1 + np.cos(radians-self.angle)) # cardioid with gain
    of 1 at front and 0 at back
        gains = self.add_noise_to_signal(gains, noise_level)

        if graph:
```

```
            plt.polar(radians, gains)
            plt.polar(source_angle, 1, 'o')
            plt.show()

        # get index of gains corresponding to DOA
        angle_index = np.abs(radians - source_angle).argmin()

        # calculate distance
        distance = ((ypos-self.ypos)**2 + (xpos-self.xpos)**2) **.5
        if graph:
            print('distance', distance)

        # scale gain by distance, 1/r^2
        dist_coeff = 1/(distance**2)
        if graph:
            print('dist_coeff', dist_coeff)
        gains*=dist_coeff

        return gains[angle_index]

    def getDirectionFromGains(self, gain1, gain2, mic2, graph = True): #
returns direction in radians of source
        # assumes angle between microphones is pi/2
        # calculate gain ratios between 2 cardioids at pi/2
        gains1, radians = self.getAngleGainVectors()
        gains2, radians = mic2.getAngleGainVectors()
        xy_gain_ratios = (gains1/gains2)[:180]

        # get index of gain ratios
        index = np.abs(xy_gain_ratios - gain1/gain2).argmin()
        angle = index*np.pi/180

        #plotting
        if graph:

            plt.polar(radians, gains1)
            plt.polar(radians, gains2)
            #plt.polar(radians[:180],xy_gain_ratios/5)
            plt.polar(angle % (2*np.pi),1,'o')
            plt.title('Dot for estimated location')
            plt.show()

        #index is angle in degrees
        return angle
```

```
83
84      def getDistance(self, angle, gain):
85          # given gain and angle, return the distance from the source
86          mic_gains, radians = self.getAngleGainVectors()
87
88          # turn angle into index(degrees) for gain
89          index = np.abs(radians-angle).argmin()
90
91          # reverse of mic_gain = gain * (1/r**2) (calculate r)
92          r = ( mic_gains[index] / gain ) ** .5
93
94          return r
95
96      def introduceDiscretizationError(self, gain, lookup_table):
97          # Simulate discretization error by finding the closest value in
    the lookup table
98          quantized_gain = lookup_table[np.abs(lookup_table - gain).argmin()
    ]
99          return quantized_gain
100
101     def adjust_angle(self, optimal_angle):
102         """
103         Adjusts the microphone's angle to the optimal_angle calculated.
104         """
105         self.angle = optimal_angle % (2 * np.pi)
106
107     # Function to plot microphones in cardioid pattern in the workspace
108     def plot_initial_microphones(mics):
109         plt.figure(figsize=(8, 8))
110
111         for i, mic in enumerate(mics):
112             # Plot the microphone's original orientation with a cardioid
    pattern
113             angles = np.linspace(0, 2 * np.pi, 100)
114             cardioid_x = 0.5 * (1 + np.cos(angles)) * np.cos(angles + mic.
    angle) + mic.xpos
115             cardioid_y = 0.5 * (1 + np.cos(angles)) * np.sin(angles + mic.
    angle) + mic.ypos
116             plt.plot(cardioid_x, cardioid_y, label=f'{mic.name} Original
    Pattern')
117
118         plt.xlim(-4, 7)
119         plt.ylim(-4, 7)
120         plt.xlabel('X position')
```

```
        plt.ylabel('Y position')
        plt.title('Initial Microphone Configuration')
        plt.grid(True)
        plt.legend()
        plt.show()


--------------------------------------------------
def optimize_sensor_angles(mics, sources):
    """
    Optimizes sensor by picking the closest source to each microphone and adjusting the angle of
    the microphone to place the source at a 45-degree angle from the microphone's axis, maintaining a 90-degree XY configuration.
    """
    for i, mic in enumerate(mics):
        # Find the nearest source to the microphone
        nearest_source = min(sources, key=lambda s: np.hypot(s[0] - mic.xpos, s[1] - mic.ypos))
        # Calculate the angle to the nearest source
        source_angle = np.arctan2(nearest_source[1] - mic.ypos, nearest_source[0] - mic.xpos)

        # Adjust the microphone angle to place the source at a 45-degree angle
        # from the microphone's axis, maintaining a 90-degree XY configuration.
        if i % 2 == 0:  # Assuming the first mic is on the left
            optimal_angle = source_angle - np.pi/4
        else:  # Assuming the second mic is on the right
            optimal_angle = source_angle + np.pi/4

        # Adjust the microphone angle
        mic.adjust_angle(optimal_angle)
        rotation_needed = source_angle - np.pi/4

        # Print the source angle and the optimal angle for each microphone
        print(f"Mic {i+1}: Source Angle = {np.degrees(source_angle):.2f} degrees, Rotation Needed = {np.degrees(rotation_needed):.2f} degrees")

# Note: Ensure that the Mic class has a method 'adjust_angle' to change the angle attribute.


```

```
157 ------------------------------------------------
158 def simulate_environment(mics, source, index, noise_level=0):
159     plt.figure(figsize=(8, 8))
160 
161     # Calculate midpoint for the 45-degree line between mics if they are
    positioned at the same point
162     midpoint = (0, 0)  # Setting the midpoint to (0, 0) for overlapping
    cardioid patterns
163 
164     # simulate microphone gains
165     gain1 = mics[0].getGainFromSource(source[0],source[1], graph=False)
166     gain2 = mics[1].getGainFromSource(source[0],source[1], graph=False)
167 
168     # calculate sound source location using microphone gains
169     source_angle =  mic1.getDirectionFromGains(gain1,gain2,mic2, graph=
    False)
170 
171     for i, mic in enumerate(mics):
172 
173         # Adjust the microphone angle to maintain a 45-degree angle with
    the source
174         mic.angle = source_angle + (-1)**i * np.radians(45)  # Alternate
    between adding and subtracting 45 degrees
175 
176         # Draw a line representing the 45-degree angle between the
    microphone and the source
177         line_45_length = 1.5  # Length of the 45-degree line
178         angle_45 = source_angle + (-1)**i * np.radians(45)
179         plt.plot([mic.xpos, mic.xpos + line_45_length * np.cos(angle_45)],
180                  [mic.ypos, mic.ypos + line_45_length * np.sin(angle_45)],
181                  'r--', label=f'Fixed Degree Line Mic {i+1}')
182 
183         # Plot the microphone's orientation with a cardioid pattern
184         angles = np.linspace(0, 2 * np.pi, 100)
185         cardioid_x = 0.5 * (1 + np.cos(angles)) * np.cos(angles + mic.
    angle) + midpoint[0]
186         cardioid_y = 0.5 * (1 + np.cos(angles)) * np.sin(angles + mic.
    angle) + midpoint[1]
187         plt.plot(cardioid_x, cardioid_y, label=f'Mic {i+1} Pattern')
188 
189         # Draw a line representing the 0-degree (direct) angle to the
    nearest source
190         line_0_length = 2.5  # Length of the 0-degree line
```

```
        plt.plot([mic.xpos, mic.xpos + line_0_length * np.cos(source_angle)],
                 [mic.ypos, mic.ypos + line_0_length * np.sin(source_angle)],
                 'b-', label=f'Optimization Line' if i == 0 else '')

    # Label the microphones with padding
    label_padding_x = 1.2
    label_padding_y = 1.2
    plt.text(mics[0].xpos - label_padding_x, mics[0].ypos + label_padding_y, 'Mic 1', fontsize=12, ha='left', va='bottom')
    plt.text(mics[1].xpos + label_padding_x, mics[1].ypos - label_padding_y, 'Mic 2', fontsize=12, ha='right', va='bottom')

    # Plot the positions of the source
    #for i, source in enumerate(source):
    plt.plot(source[0],source[1], 'o', label='Source')

    plt.legend()
    plt.xlim(-4, 7)
    plt.ylim(-4, 7)
    plt.xlabel('X position')
    plt.ylabel('Y position')
    plt.title('Simulation Environment')
    plt.grid(True)
    plt.savefig(os.getcwd()+"/file%02d.png" % index)
    plt.close()

-----------------------------------------------
#GENERATE VIDEO
# Run to turn saved figures into a video
subprocess.call([
    'ffmpeg', '-framerate', '8', '-i', 'file%02d.png', '-r', '30', '-pix_fmt', 'yuv420p',
    'target_tracking.mp4'
])
for file_name in glob.glob("*.png"):
    os.remove(file_name)

-----------------------------------------------
EXAMPLE USE OF GETGAINFROMSOURCE
# mic is at the origin with an angle of 0
mic1 = mic('mic1',0,0,np.pi/2)

```

```
gain = mic1.getGainFromSource(1, 0) # source is at position x=0,y=1
print('gain:',gain)

-----------------------------------------------
EXAMPLE USE OF GETDIRECTIONFROMGAINS
mic1 = mic('mic1',0,0,3*np.pi/4)
mic2 = mic('mic2',0,0,np.pi/4)

# arbitrary sound source location
xpos = 1
ypos = 0

# simulation: calculate the signal gains given the source location
gain1 = mic1.getGainFromSource(xpos,ypos,graph=False)
gain2 = mic2.getGainFromSource(xpos,ypos,graph=False)

# prediction: estimate the sound source angle given the signal gains
angle =  mic1.getDirectionFromGains(gain1,gain2,mic2)
print("Estimated angle of source in radians:",angle)

# prediction: estimate the sounde source distance given the angle and
   source loudness
dist = mic1.getDistance(angle,gain1)
print('Estimated distance of source:',dist)

-----------------------------------------------
EXAMPLE USE OF INTORDUCEDISCRETIZATIONERROR
# mic is at the origin with an angle of 0
mic1 = mic('mic1', 0, 0, np.pi/2)

# Define a lookup table for gain values
resolution = 1000  # Adjust the resolution as needed
lookup_table = np.linspace(0, 1, resolution)

# Simulate discretization error for gain using the lookup table
gain = mic1.getGainFromSource(1, 0)  # Replace with actual values
quantized_gain = mic1.introduceDiscretizationError(gain, lookup_table)
print('Original gain:', gain)
print('Quantized gain:', quantized_gain)

# Assume the mic class and the optimize_sensor_angles function have been
   defined as discussed

-----------------------------------------------
```

```
EXAMPLE USE OF SIMULATE_ENVIRONMENT AND OPTIMIZATION
# Simulation Example

# Create microphones, mic1 being the left and mic2 being the right in the XY configuration
mic1 = mic('Mic1', 0, 0, np.pi/2)  # Initial angle set to 0 degrees (pointing upwards)
mic2 = mic('Mic2', 0, 0, 0)  # Both mics are assumed to be at the same point for XY pair

# Plot initial microphone configuration
mic.plot_initial_microphones([mic1, mic2])

# Define sources
sources = [(1.5, 4), (-2, 6)]

# Optimize microphone angles based on sources
optimize_sensor_angles([mic1, mic2], sources)

# Simulate environment showing the optimized angles
simulate_environment([mic1, mic2], sources)
```