Report on 3D Source Localization Using Time Difference of Arrival

By: Daniel Plotkin & Beniamin Borowski

EENG 320

NYIT ID: 1307987, 1311550  
Date: 12/3/2024

**Table of Contents:**

1. Objective……………………………………………………………………………………………………… pg. 3
2. Procedure……………………………………………………………………………………………………. pg. 3
3. Block Diagram……………………………………………………………………………………………… pg. 4
4. Results………………………………………………………………………………………………………… pg. 4
5. Appendix A: Finding the Predicted TDoA………………………………………………………. pg. 5
6. Appendix B: Defining the Error Function………………………………………………………. pg. 6
7. Appendix C: Trust Region Reflexive Algorithm……………………………………………… pg. 6
8. Appendix D: Camera Direction……………………………………………………………………... pg. 7
9. Appendix E: Group Contributions…………………………………………………………………. pg. 8
10. Appendix F: Code…………………………………………………………………………………………. pg. 8

**Objective:**

The goal of this project is to estimate the position of a sound source in a 3D space using Time Difference of Arrival (TDoA) measurements and to then direct a camera to point towards the location of the source of the sound.

**Procedure:**

1. TDoA Simulation:

Time Difference of Arrival (TDoA) is simulated based on the known positions of microphones and the true sound source. The TDoA values are calculated as the relative differences in arrival times of sound at each microphone. One microphone is selected to act as the reference microphone, i.e. the time of arrival of that microphone is always set to 0. Then the arrival time of the other microphones are used to find the relative differences between the time of arrivals. Normally, in a non-simulated version of this system, the input times from the microphones would not need to be generated, since a physical sound will actually be present. The location of the true sound source is not used again in any part of the calculation, except to compare the accuracy of the results. (*See Appendix A*)

2. Source Localization (Nonlinear Least Squares):

Using the observed TDoA values(the ones that would be physically obtained by the microphones), the sound source position is estimated by minimizing the error between the observed and predicted TDoA values. At first, the code assumes that the sound sources location is in the center of the microphone array. Then it calculates what the TDoA values for that given location would be (since the speed of sound is a known quantity). An error function is generated between the observed TDoA values & the predicted TDoA values for the chosen location (*see Appendix B*). This function is then minimized (*see Appendix C*), and the output provides the location of the noise source.

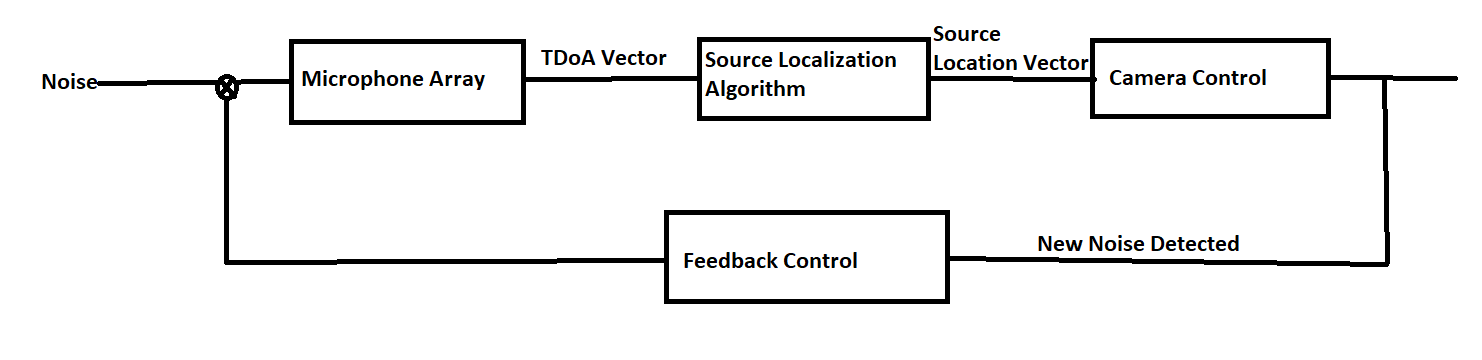
3. Camera Direction Calculation:

Once the estimated position of the source is obtained, the direction for a camera to face the source is computed. The direction vector is normalized, and the angles in the XY plane and elevation are calculated. This provides the camera orientation required to face the source. (*See Appendix D*).

4. Visualization:

A 3D plot is generated to visualize the positions of the microphones, the true and estimated source positions, and the camera. The camera's direction vector is represented using a quiver plot, allowing for clear depiction of its orientation.

**Block Diagram:**

****

This block diagram illustrates the operation of a dynamic noise detection and localization system. The process begins with noise being captured by a microphone array, which measures the time differences at which the sound reaches each microphone. These time differences are packaged into a Time Difference of Arrival (TDoA) vector, providing crucial data for pinpointing the noise source.

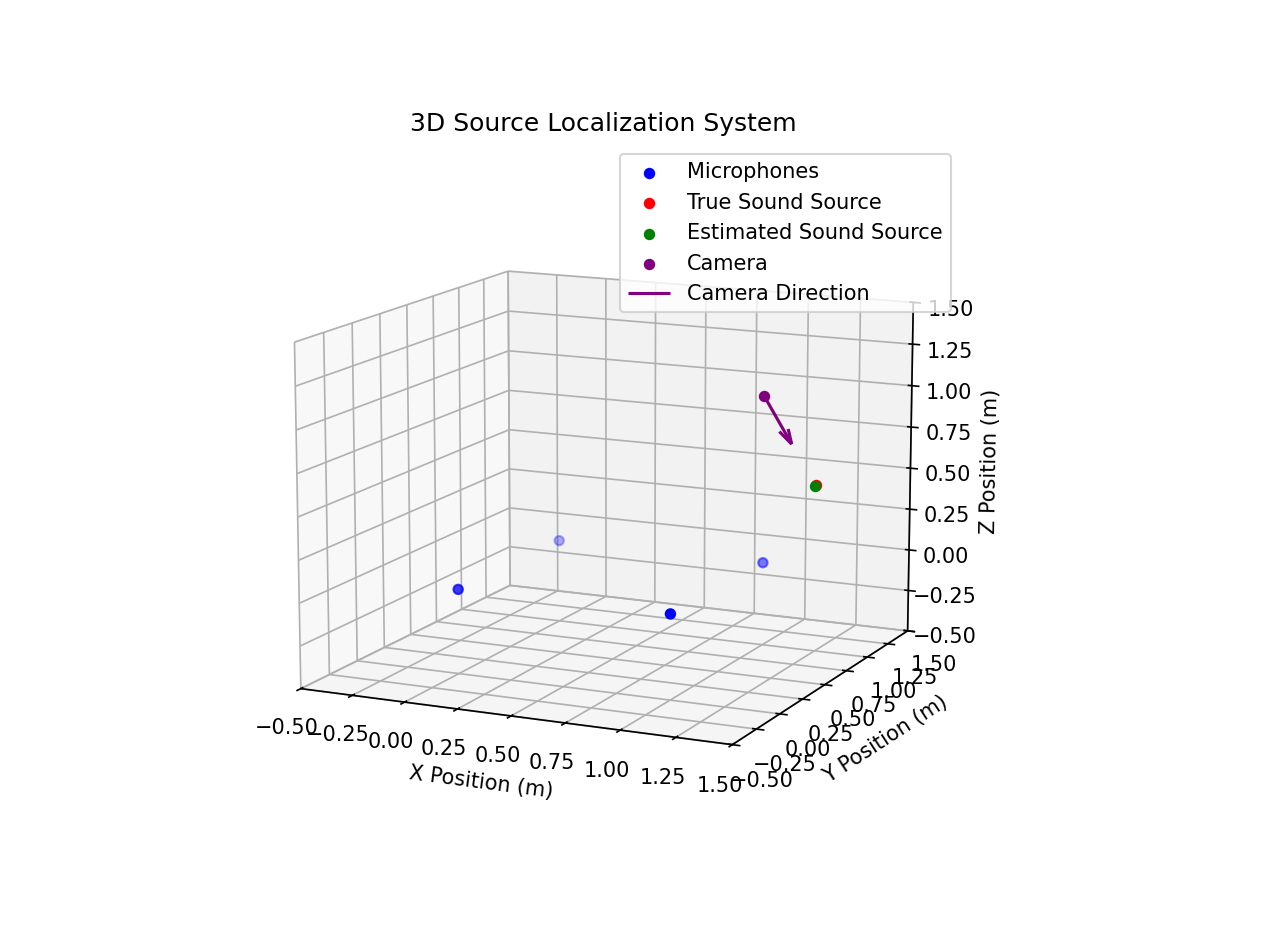
The TDoA vector is then processed by the Source Localization Algorithm, which calculates the precise location of the noise source. The resulting source location vector is sent to the camera control system, which adjusts the camera's orientation to focus on the noise source, enabling real-time monitoring.

What makes this system particularly powerful is its feedback control loop. If new noise is detected by the microphone array, the system dynamically reinitializes. The feedback loop ensures the camera and localization algorithm stay responsive, adapting to new noise sources in the environment. (Implemented in the physical version of this system, not in this simulated one).

**Results:**

True Sound Source Position: [1.25 1. 0.5 ]

Estimated Source Position: [1.24634335 0.99770581 0.49734931]

Camera XY Angle (degrees): -0.5335779493058805Camera Elevation Angle (degrees): -63.89007316361392  
  


**Appendix A**   
Finding the predicted TDoA

Where denotes the position of the microphone in the array (in the simulated case provided in appendix , 4 microphones are used).

Note: is the location of the reference microphone

Where denotes the estimated position of the noise source.

Where denotes the distance between the estimated position of the source & the microphone.

Note:

Where denotes the predicted TDoA from the estimated position of the sound source.

**Appendix B**

Defining the Error Function

Where is the error function for the TDoA of microphone & the predicted source location.

The overall error vector can be defined as:

is the predicted TDoA.

Then, a function is defined:

This function , once minimized, will provide the location of the source of the sound, up to a python defined tolerance (). For this report, was minimized as follows:

least\_squares(tdoa\_error, initial\_guess)

This code uses scipy.optimize.least\_squares(a,b) where a is the function being minimized & b is the initial guess. The code iteratively updates the estimated position vector, in order to minimize .  
Note: The two lines of code highlighted in red are equivalent.

**Appendix C**

Trust Region Reflexive Algorithm

In the case of this report, the algorithm used for minimizing is called the Trust Region Reflexive Algorithm(TRF)**.**

Key Steps of the Algorithm:

1. Trust Region Definition:
   * At each iteration, the algorithm defines a "trust region" around the current estimate of the parameters. This region represents the area where the local quadratic approximation of the objective function is considered reliable.
2. Objective Function:
   * The TRF algorithm minimizes the sum of squared residuals. In our case, this function can be expressed as:
   * Note: the is included as convention in order to make finding the gradient easier. It is a constant that does not affect the final output.
3. Linear Approximation:

* Within the “trust region”, the algorithm approximates a linear version of the error vector:
* For small changes in the estimated position:

is the Jacobian matrix of the residuals, defined as:

* The algorithm then iteratively solves for

1. Trust Region Adjustment:
   * After each iteration, the trust region radius is updated based on the accuracy of the quadratic model:
     + If the model is accurate, the trust region is expanded.
     + If the step leads to a poor reduction in the objective function, the trust region is reduced.

**Appendix D**

Camera Direction

1. The direction from the cameras position to the source is calculated as follows:

Where:

* is the position vector describing the noise source location.
* is the position vector describing the camera’s current position.

2. Normalization (Unit Vector)

To get the direction vector as a unit vector (i.e., a vector that only represents direction and not magnitude), the vector is normalized.

3. XY Plane Angle

The first angle to compute is the angle between the direction vector and the positive X-axis in the horizontal (XY) plane.

The function arctan2 returns the angle in radians accounting for the signs of both components to give the correct quadrant of the angle.

4. Elevation Angle (Pitch Angle)

The second angle is the elevation angle, which is the angle between the direction vector and the XY plane. This represents the vertical direction the camera needs to face.

**Appendix E**

Group Contributions

Beniamin Borowski: Block Diagram, ReadMe

Daniel Plotkin: Coding, Report

**Appendix F**

Code:  
import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import least\_squares

#  Step 1: Simulate TDoA

def simulate\_tdoa(mic\_positions, source\_position, speed\_of\_sound):

    """

    Simulate TDoA values given microphone positions and the true source position.

    """

    distances = np.linalg.norm(mic\_positions - source\_position, axis=1)

    arrival\_times = distances / speed\_of\_sound

    tdoa = arrival\_times - arrival\_times[0]  # Relative to the first mic

    return tdoa

#  Step 2: Source Localization (Nonlinear Least Squares)

def TDoA\_Least\_Squares(mic\_positions, tdoa, speed\_of\_sound):

    """

    Estimate the source position using nonlinear least squares to minimize TDoA error.

    """

    def tdoa\_error(estimated\_position):

        # Compute predicted distances and TDoAs in 3D

        distances = np.linalg.norm(mic\_positions - estimated\_position, axis=1)

        predicted\_tdoa = (distances - distances[0]) / speed\_of\_sound

        return tdoa - predicted\_tdoa

    # Initial guess: center of the microphone array

    initial\_guess = np.mean(mic\_positions, axis=0)

    # Perform least squares

    result = least\_squares(tdoa\_error, initial\_guess)

    return result.x  # Optimized position

#  Step 3: Calculate Camera Direction (3D)

def calculate\_camera\_direction(source\_position, camera\_position):

    """

    Calculate the 3D direction for the camera to face the sound source.

    """

    direction\_vector = source\_position - camera\_position

    norm = np.linalg.norm(direction\_vector)

    direction\_unit = direction\_vector / norm  # Normalize the vector to get direction

    # Calculate XY and elevation angles

    XY = np.degrees(np.arctan2(direction\_unit[1], direction\_unit[0]))  # XY plane

    elevation = np.degrees(np.arcsin(direction\_unit[2]))  # Z direction

    return direction\_unit, XY, elevation

#  Step 4: Visualization (3D Plot)

def plot\_system(mic\_positions, true\_source, estimated\_position, camera\_position, camera\_direction, camera\_angle):

    """

    Plot the system in 3D with positions of microphones, true sound source,

    estimated position, and camera direction.

    """

    fig = plt.figure(figsize=(10, 8))

    ax = fig.add\_subplot(111, projection='3d')

    # Plot microphones, true source, and estimated position

    ax.scatter(mic\_positions[:, 0], mic\_positions[:, 1], mic\_positions[:, 2], c='blue', label='Microphones')

    ax.scatter(\*true\_source, c='red', label='True Sound Source')

    ax.scatter(\*estimated\_position, c='green', label='Estimated Sound Source')

    ax.scatter(\*camera\_position, c='purple', label='Camera')

    # Plot the camera direction in 3D using a quiver plot

    ax.quiver(camera\_position[0], camera\_position[1], camera\_position[2],

              camera\_direction[0], camera\_direction[1], camera\_direction[2],

              color='purple', length=0.3, label='Camera Direction')

    # Set axis limits and labels

    ax.set\_xlim(-0.5, 1.5)

    ax.set\_ylim(-0.5, 1.5)

    ax.set\_zlim(-0.5, 1.5)

    ax.set\_xlabel("X Position (m)")

    ax.set\_ylabel("Y Position (m)")

    ax.set\_zlabel("Z Position (m)")

    ax.grid()

    ax.legend()

    ax.set\_title("3D Source Localization System")

    plt.show()

#  Main Execution

# Microphone positions (in meters) now in 3D

mic\_positions = np.array([

    [0, 0, 0],  # Bottom-left corner

    [1, 0, 0],  # Bottom-right corner

    [0, 1, 0],  # Top-left corner

    [1, 1, 0]   # Top-right corner

])

# True sound source position (in 3D, for simulation/testing only)

true\_source = np.array([1.25, 1.0, 0.5])  # True position (unknown in real use)

# Speed of sound (in meters per second)

speed\_of\_sound = 343

# Simulate TDoA values

observed\_tdoa = simulate\_tdoa(mic\_positions, true\_source, speed\_of\_sound)

# Perform localization

estimated\_position =  TDoA\_Least\_Squares(mic\_positions, observed\_tdoa, speed\_of\_sound)

# Set camera position

camera\_position = [1,1,1]

# Calculate the camera's 3D direction vector and angles

camera\_direction, camera\_angle\_XY, camera\_angle\_elevation = calculate\_camera\_direction(estimated\_position, camera\_position)

# Print Results

print(f"True Sound Source Position: {true\_source}")

print(f"Estimated Source Position: {estimated\_position}")

print(f"Camera XY Angle (degrees): {camera\_angle\_XY}")

print(f"Camera Elevation Angle (degrees): {camera\_angle\_elevation}")

# Visualize the system in 3D

plot\_system(mic\_positions, true\_source, estimated\_position, camera\_position, camera\_direction, camera\_angle\_XY)