

design a 2d system for triangulation of position detection system based on 3 microphones each giving a trigger signal when a sound is received. Please code in c

Below is a **complete 2D acoustic triangulation (more precisely: multilateration)** system design using **3 microphones**, each producing a **trigger timestamp** when a sound is detected. The position of the sound source is computed from **time-difference-of-arrival (TDOA)**.

This is a **standard, robust approach** used in embedded systems, robotics, and acoustic localization.

1. System Overview

Assumptions

- 3 microphones at **known fixed positions**
 - All microphones are **time-synchronized**
 - Each microphone outputs a **timestamp** when sound crosses a threshold
 - Sound propagates at constant speed c (≈ 343 m/s at 20°C)
 - Sound source is in a **2D plane**
-

2. Geometry & Math Model

Microphone Positions

Let:

yaml

 Copy code

Mic 1: (x_1, y_1)

Mic 2: (x_2, y_2)

Mic 3: (x_3, y_3)

Trigger Times

scss

 Copy code

t_1, t_2, t_3 (seconds)

Distance to Source

ini

 Copy code

$d_1 = c * t_1$

$d_2 = c * t_2$

$d_3 = c * t_3$

We don't know absolute emission time, so we use **time differences**:

r

 Copy code

$$\Delta t_{21} = t_2 - t_1 \rightarrow \Delta d_{21} = c * \Delta t_{21}$$

$$\Delta t_{31} = t_3 - t_1 \rightarrow \Delta d_{31} = c * \Delta t_{31}$$

This produces **two hyperbolas** whose intersection gives the source position.

3. Linearized Solution (Efficient for Embedded Systems)

Subtract squared distance equations:

csharp

 Copy code

$$(x - x_2)^2 + (y - y_2)^2 - (x - x_1)^2 - (y - y_1)^2 = \Delta d_{21}^2 + 2\Delta d_{21} d_1$$

$$(x - x_3)^2 + (y - y_3)^2 - (x - x_1)^2 - (y - y_1)^2 = \Delta d_{31}^2 + 2\Delta d_{31} d_1$$

This reduces to a **2x2 linear system**:

java

 Copy code

$$A x = b$$

Solved with standard linear algebra.

4. C Implementation

Features

- ✓ No dynamic memory

- ✓ Suitable for MCU / embedded
 - ✓ Numerically stable
 - ✓ Clear structure
-

C Code (Complete)

c

 Copy code

```
#include <stdio.h>
#include <math.h>

#define SPEED_OF_SOUND 343.0    // meters per second

typedef struct {
    double x;
    double y;
} Point2D;

/*
 * Solve 2D sound source localization using 3 microphones
 * Inputs:
 *   mic[] : microphone positions (size 3)
 *   t[]   : trigger times in seconds (size 3)
 * Output:
 *   src   : estimated sound source position
 * Returns:
 *   0 on success, -1 on failure (singular geometry)
 */
```

```

*/
int localize_sound_2d(Point2D mic[3], double t[3], Point2D *src)
{
    // Time differences
    double dt21 = t[1] - t[0];
    double dt31 = t[2] - t[0];

    // Distance differences
    double dd21 = SPEED_OF_SOUND * dt21;
    double dd31 = SPEED_OF_SOUND * dt31;

    // Matrix coefficients
    double A11 = 2 * (mic[1].x - mic[0].x);
    double A12 = 2 * (mic[1].y - mic[0].y);
    double A21 = 2 * (mic[2].x - mic[0].x);
    double A22 = 2 * (mic[2].y - mic[0].y);

    double B1 = dd21 * dd21
        - (mic[1].x * mic[1].x + mic[1].y * mic[1].y)
        + (mic[0].x * mic[0].x + mic[0].y * mic[0].y);

    double B2 = dd31 * dd31
        - (mic[2].x * mic[2].x + mic[2].y * mic[2].y)
        + (mic[0].x * mic[0].x + mic[0].y * mic[0].y);

    // Determinant
    double det = A11 * A22 - A12 * A21;

    if (fabs(det) < 1e-6) {

```

```

        return -1; // microphones are collinear or geometry invalid
    }

    // Solve linear system
    src->x = (B1 * A22 - B2 * A12) / det;
    src->y = (A11 * B2 - A21 * B1) / det;

    return 0;
}

/* Example usage */
int main(void)
{
    Point2D microphones[3] = {
        {0.0, 0.0},
        {2.0, 0.0},
        {0.0, 2.0}
    };

    double times[3] = {
        0.0123,
        0.0131,
        0.0128
    };

    Point2D source;

    if (localize_sound_2d(microphones, times, &source) == 0) {
        printf("Sound source at (%.3f, %.3f)\n", source.x, source.y);
    }
}

```

```
    } else {  
        printf("Localization failed\n");  
    }  
  
    return 0;  
}
```

5. Practical Design Notes

Hardware

- Use **hardware timers** for timestamp capture
- Use **interrupts** for microphone triggers
- Prefer **high-resolution timers** (≥ 1 MHz)

Accuracy Improvements

- Apply **band-pass filtering**
 - Use **cross-correlation** instead of threshold triggers
 - Average multiple detections
 - Calibrate speed of sound using temperature sensor
-

6. Limitations

- 3 microphones → **one unique solution in 2D**
- Collinear microphone placement will **fail**

- Noise and echoes degrade accuracy
-

If you want, I can:

- Add **noise-robust least squares**
- Convert this to **fixed-point math**
- Extend to **4+ microphones**
- Provide a **microcontroller-ready ISR design**
- Add **cross-correlation TDOA** instead of triggers

Just tell me.

