Acoustic Source Localization Techniques

Team Spectrum

Vishva Bhate
[EDM18B54]

Mayank N. Mehta
[EDM18037]

Abstract

This project aims at building a modular simulation tool to emulate the functionality of a passive SONAR system in terrestrial environments. Two algorithms are described in the project which locate an audio source. One of the algorithms works in 2D while the other is built for 3D. The results of the algorithms are compared with the actual values and the efficiency and limitation of the algorithms are presented. The simulation tool built as part of this project can be used during the design phase of a passive SONAR system.

Contents

Chapter 1

Introduction

This chapter discusses about

- motivation to do the project and the objective
- theory relevant to the project
  - random signals
  - narrow and broad band signals
  - the physical pre-requisites
  - Nyquist-Shannon sampling theorem's application in this project
  - near and far-field conditions
  - microphones
  - environmental attenuation
- assumptions about the system made in this project

1.1 Motivation

With the proliferation of marine vessels for trade and military applications, the safety of territorial waters poses new challenges to the nation's navy. Tracking foreign vessels without being exposed and alerting them is a critical challenge. Although active SONAR techniques exist which help in monitoring the surroundings, it has a major drawback of exposing the tracker to the tracked. Passive SONAR technique is employed to achieve the objective of stealth tracking. We wish to explore how this technology works, understand the limitations and build a better design in the future.

1.2 Objective

Our objective is to build a simulation tool to test acoustic source localization algorithms in terrestrial and underwater environments.

We had planned to build both the hardware and the software as a part of the project. But, with the sudden outbreak of the SARS-CoV-2 in the country and declaration of multiple lockdowns, we were forced to modify our plans and stick to simulation.

1.3 Random Signals

Note: *This section aims to provide an overview of the text mentioned in Appendix A of [1] which is relevant in the context of this project.*

Random Process A random process is an indexed family of random variables $\{x_n\}$ characterized by a set of probability distribution functions that, in general, may be a function of the index n.
In using the concept of a random process as a model for discrete-time signals, the index n is associated with the time index. In other words, each sample value $x[n]$ of a random signal is assumed to have resulted from a mechanism that is governed by a probability law. A random process in which the probability distributions of random variables are independent of a shift of time origin is called a **stationary process**.

In many applications of discrete-time signal processing, random process serve as models for signals in the sense that a particular signal can be considered a sample sequence of a random process. Although the details of such signals are unpredictable, *(which makes taking a deterministic approach to signal representation inappropriate)* certain average properties like expectation, variance, etc. can be determined, given the probability law of the process. These properties do not completely characterize such signals.

Processes for the expectation and the variance of a random variable are constant and independent of n called **wide-sense stationary process (WSS)**.

Cross-correlation A measure of the dependence between two different WSS random signals is obtained from the cross-correlation sequence. If $\{x[n]\}$ and $\{y[n]\}$ are two random processes, their cross-correlation is given by

$$\phi_{xy}[m] = \mathcal{E}[x[n+m]y^*[n]] \tag{1.1}$$

where $\mathcal{E}[\cdot]$ represents expectation.

Application of the concepts in the project

- The source signal to be localized is modelled as a WSS in most of the literature on the topic of acoustic source localization.
- The statistical definition of cross correlation mentioned above is used widely in TDoA estimation.

1.4 Narrowband vs Broadband signals

Broadband signal In the context of acoustic source localization, we refer to the source signal as a broadband signal if the energy of the signal is distributed amongst a wide range of frequencies. For example, if we are dealing with localization of human speech, we refer to the source signal as broadband signal as human voice consists of frequency components ranging from 20 Hz to 20 kHz.

Narrowband signal In the context of acoustic source localization, we refer to the source signal as a narrowband signal if the energy of the signal is not distributed between multiple frequencies but concentrated around a centre frequency.

Application of the concepts in the project

In this project, the source is assumed to emit a single frequency signal - either sine or cosine. But, we know that generation of a pure sine or cosine wave is not possible. We always have minor distortions in the signal. Hence, a frequency analysis of such real-life signal shows that energy of the signal is concentrated in it's centre frequency. Therefore, we will be dealing with narrowband signals in this project.

1.5 Propagating waves and sampling

Sound is a vibration that propagates as an acoustic wave, through a transmission medium such as a gas, liquid or solid. A sound wave is basically a sequence of successive compressions and rarefactions in which the direction of oscillation of the particles and the wave are parallel. The physics of this propagation is described by wave equation for the appropriate medium and boundary conditions. The wave equation, in Cartesian coordinate system, is given as:

$$\frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} + \frac{\partial^2 s}{\partial z^2} = \frac{1}{c^2}\frac{\partial^2 s}{\partial t^2}$$

where, $s(\vec{x}, t)$ represents a general scalar field and c is the speed of sound. The solution to the wave equation is of the form:

$$s(x, y, z, t) = Ae^{j(\omega t - k_x x - k_y y - k_z z)}$$

where A is a complex constant and k_x, k_y, k_z and ω are real constants with $\omega \geq 0$. This solution is referred as a *monochromatic plane wave*.

One thing which is clear from above discussion is that a sound wave is a **spatio-temporal wave**[2] Hence, to obtain all the information present in the wave, the sensors must sample the wave both spatially as well as temporally in accordance to the Nyquist-Shannon sampling theorem, to eliminate aliasing. Hence, for a band-limited wave signal:

- By application of sampling theorem in time domain, if the maximum frequency component in the wave signal is f_{max}, then, the sensors must sample at a rate of

  $$f_s \geq 2f_{max} \tag{1.2}$$

  where f_s is the **sampling frequency** of each sensor in the array.

- By application of sampling theorem in spatial domain, the maximum **spatial frequency**, d, or the distance between two sensors must satisfy the following relationship:

  $$d \leq \frac{\lambda_{min}}{2} \tag{1.3}$$

  where λ_{min} is minimum wavelength component present in the source wave signal.

Application of the concepts in the project

- The source signal propagating through space needs to be sampled via sensors to process the signal. The constraints imposed by the sampling theorems help in obtaining non-aliased sequences for processing.

1.6 Near field and far field

When we observe the sound wave front emanating from a point source from the viewpoint of the sensor, two cases arise:

1. Near field
2. Far field

We say the sensor is in the **far field** when the sensor is at significant distance from the sound wave source. As the curved wave form travels, it becomes more spread out across the normal of the direction of travel. This effect causes the wave shape to become more planar. This region is represented by:

$$R > \frac{2d^2}{\lambda} \tag{1.4}$$

where R is the radial distance between the source and sensor, d is distance between two sensors and λ is the wavelength of the acoustic wave. For distances that are less than this condition, the sensor is said to be located in the **near field**, where the waveform still retains curvature in its shape. As the waveform moves further, the curvature is reduced[2].

Application of the concepts in this project

- The algorithm required to localize a source in space depends on in which field is the sensor placed.
- A far field simplifies the approach to localization.

1.7 Short note on microphones

A microphone is an electro-acoustic transducer receptor which translates acoustic signals into electrical signals. A microphone *(or commonly known as mic)*, has two different parts:

1. **Mechanical Acoustic Transducer (MAT)**: turns pressure variations in air into vibrations of a mobile element called a diaphragm.
2. **Electric Mechanical Transducer (EMT)**: converts vibrations from MAT into voltage and electric current.

Microphones are distinguished based on directivity. **Directivity** is the characteristic of a microphone which describes what would be the output of microphone according to the angle of incidence of the input wave. Three most common microphone patterns are:

- **Omnidirectional**: Microphone delivers same electrical output independently of angle of incidence.

- **Bidirectional**: Captures sound coming from the front and rear of the microphone but not from the sides.

- **Cardiod**: They are unidirectional microphones having a heart shaped polar pattern. Their sensitivity is higher for sounds incident from the front.

1.8 Environmental Attenuation

When a sound wave travels through a medium, its intensity diminishes with distance. The diminishing occurs due to of loss of mechanical energy of vibration due to properties of the medium of travel like viscosity, density, etc. In terrestrial environment, atmospheric parameters like pressure, relative humidity, and temperature also play a vital role as they influence the medium's (air) properties. The speed of sound is also influenced due to these environmental parameters. The basic physical laws governing these effects are:

1. Sutherland's Law of Viscosity

2. Herman Wobus' Equation

3. Gas Laws

Application of the concepts in this project

In order to better simulate a terrestrial environment, we have implemented all the above physical laws to generate attenuation parameters for the signals received by the sensors and to estimate the speed of sound in given environmental conditions. The book *Fundamental of Atmospheric Physics* [3] has been referred for implementation.

1.9 Assumptions

In this project, we have made the following assumptions about the acoustic wave, the environment of propagation, and the sensors:

1. The source of the wave is assumed to be a point source undergoing wide-sense stationary process and emanating narrowband signal *ideally a pure cosine wave.*

2. The sensor array lies in the far field region on the source.

3. The sound waves passing through the sensors are monochromatic plane waves.

4. All the microphones in the array are omnidirectional.

5. The microphone and the source are present in an ideal **anechoic** chamber.

6. The source is stationary w.r.t. the sensor array.

Chapter 2

Brief Literature Survey

Extensive research has been done over the years in the domain of acoustic source localization. One of the most used technique to locate a sound source involves the usage of an array of sensors (microphones/hydrophones). One way to locate a source using an array of sensors is to find out the time difference of arrival (TDoA) the signal between two sensors. A simple cross-correlation operation could provide the desired results, but with low SNR, it fails[4]. Various *Generalized Cross-Correlation (GCC) methods*[5, 6] like *SCOT*[4], *PHAT*[7, 8] and others described by Hassab and Boucher[9] are employed. Another technique to find out the time delay makes use of the *Hilbert Transform*[10]. Non-TDoA based methods like SRP-PHAT are also used for localization[11].

The cross-correlation method, GCC-PHAT method, and Hilbert transform methods[10] of TDoA are simulated by us and discussed in the section 3.2. Along with these methods, we also present our method of TDoA estimation, and explain our final choice of estimation method.

Chapter 3

Methodology

This chapter discusses about

- calculating actual TDoA based on geometry of array
- the results of methods for estimating TDoA existing in the literature
- our method of TDoA estimation and comparision of the output
- our methods to localize the source using two geometrical configurations

3.1 Calculating actual time delay

3.1.1 Right triangle configuration

At each vertex of the triangle, a microphone is placed. Let the vertices be - P, Q, R and the right angle is at $\angle R$. We fix the coordinate system with point R as the origin. Let $(0, d_{PR})$ be the coordinates of vertex P and $(d_{QR}, 0)$ be coordinates of point Q[1].

[1] d_{PR}, d_{QR} are in accordance with equation 2.2

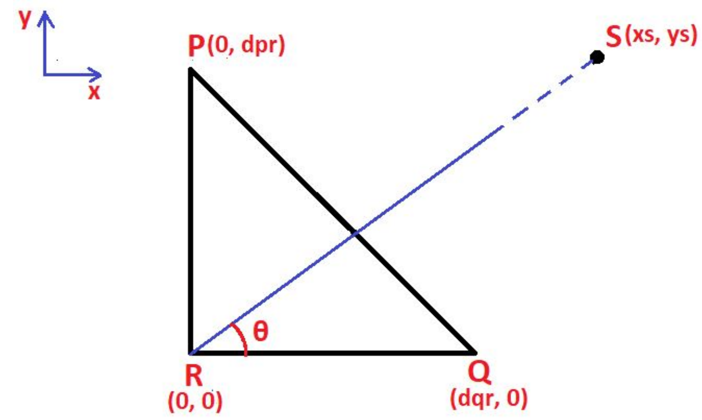


Figure 3.1: Configuration of three microphones

Let the coordinates of a point source, S, be (x_s, y_s) and the point source be very far away from the array such that the sound wavefronts can be approximated at planar *(far field condition)*. Let K, M, and N be the wavefronts passing through points Q, R, and S respectively. Perpendiculars are dropped from points Q and P onto wavefront N. Let the angle made by the propagation direction of the wave, w.r.t. `x axis` be θ.

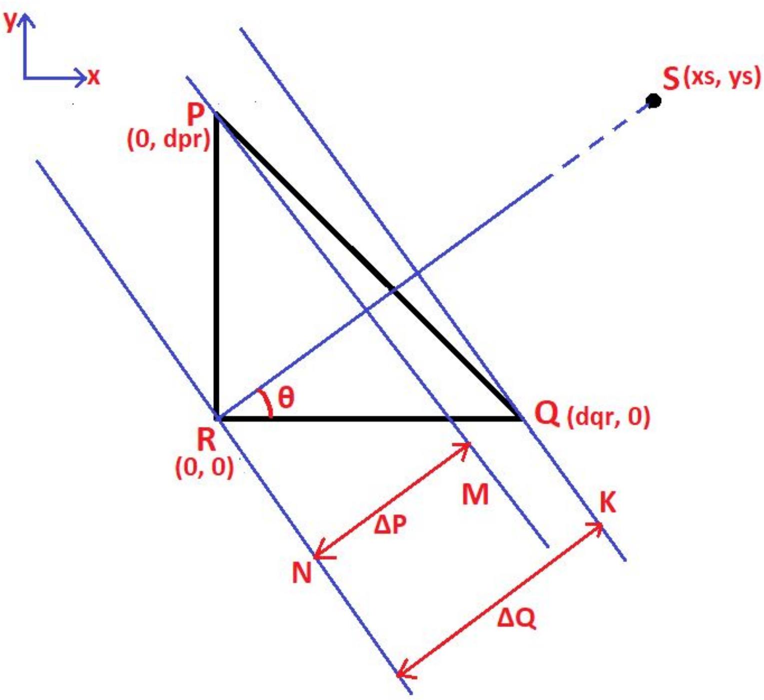


Figure 3.2: Deriving time delay

From basic trigonometry and properties of parallel lines and traversals, we can conclude the following:

$$\begin{aligned}
\angle SRQ &= \theta \\
\therefore \angle RQA &= \theta \ (\because \text{alternate interior angles are equal}) \\
\because \angle PRQ &= 90^\circ \\
\text{and } \angle PRQ &= \angle PRS + \angle SRQ \\
\implies \angle PRS &= 90^\circ - \theta \\
\therefore \angle BRP &= \theta \ (\because \Delta PRC \text{ is right triangle})
\end{aligned}$$

If ΔP is the distance between wavefronts M and N and ΔQ is distance between wavefronts K and N

$$\begin{aligned}
\Delta P &= d_{PR}\, sin(\theta) \\
\Delta Q &= d_{QR}\, cos(\theta)
\end{aligned}$$

If the speed of the sound is v m/s,

$$\begin{aligned}
\because \text{speed} &= \frac{\text{distance}}{\text{time}} \\
t_{pr} &= \frac{\Delta P}{v} \\
t_{qr} &= \frac{\Delta Q}{v}
\end{aligned}$$

where t_{pr} is the time taken for wavefront to move from M to N and t_{qr} is the time taken for wavefront to move from K to N. Hence we conclude that

$$t_{pr} = \frac{d_{PR}\, sin(\theta)}{v} \tag{3.1}$$

$$t_{qr} = \frac{d_{QR}\, cos(\theta)}{v} \tag{3.2}$$

$$\text{where, } \theta = tan^{-1}\left(\frac{y_s}{x_s}\right)$$

The time delays can also be expressed as:

$$\begin{aligned}
t_{pr} &= t_r - t_p \\
t_{qr} &= t_r - t_q
\end{aligned}$$

where t_r is the time instant when the wavefront passing through point R, t_p is the time instant when the wavefront passes through point P, and t_q is the time instant when the wavefront passes through point Q. This relation can be verified with the following examples:

- If the source is in `Quadrant-I` then, $sin(\theta) > 0 \implies t_{pr} > 0 \implies t_r > t_p$, which is valid as wavefront reaches R after it crosses P.
- If the source is in `Quadrant-II` then, $cos(\theta) < 0 \implies t_{qr} < 0 \implies t_r < t_q$, which is valid as wavefront reaches Q after it crosses R.
- If the source is in `Quadrant-III` then, $sin(\theta) < 0 \implies t_{pr} < 0 \implies t_r < t_p$, which is valid as wavefront reaches P after it crosses R.

3.1.2 Square configuration

Four microphones A, B, C, D are placed on the vertices of a square, with C being the origin and $A(0, y_p, 0)$, $B(x_q, y_q, 0)$, $D(x_r, 0, 0)$[2].

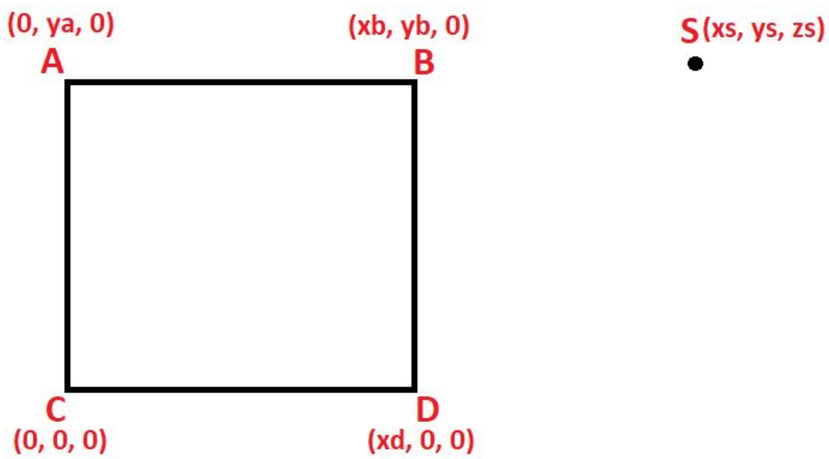


Figure 3.3: Configuration of the four microphones

Let $S(x_s, y_s, z_s)$ be the point sound source far away from the array such that the sound wave fronts can be approximated at planar (*far field condition*). Now, using the coordinates of microphone and sound source, we get the distance between them and then time difference can simply be calculated using,

$$speed = \frac{distance}{time}$$

[2] y_a, x_b, y_b, x_d are in accordance with equation 2.2

Let D_{as}, D_{bs}, D_{cs}, D_{ds}, be the distance between point source S and microphones A, B, C, D respectively and,

$$\begin{aligned} D_{as} &= \sqrt{(x_s-0)^2+(y_s-y_a)^2+(z_s-0)^2} \qquad (3.3)\\ &= vT_a \\ D_{bs} &= \sqrt{(x_s-x_b)^2+(y_s-y_b)^2+(z_s-0)^2} \qquad (3.4)\\ &= vT_b \\ D_{cs} &= \sqrt{(x_s-0)^2+(y_s-0)^2+(z_s-0)^2} \qquad (3.5)\\ &= vT_c \\ D_{ds} &= \sqrt{(x_s-x_d)^2+(y_s-0)^2+(z_s-0)^2} \qquad (3.6)\\ &= vT_d \end{aligned}$$

Where v is speed of sound and T_a, T_b, T_c, T_d, are the time taken by the sound to travel from source, S to the microphone A, B, C, D respectively. Now consider the following,

$$\begin{aligned} D_{ac} &= D_{as}-D_{cs} = v(T_a-T_c) = vt_{ac} \\ &= \sqrt{(x_s-0)^2+(y_s-y_a)^2+(z_s-0)^2} - \sqrt{(x_s-0)^2+(y_s-0)^2+(z_s-0)^2} \end{aligned} \qquad (3.7)$$

$$\begin{aligned} D_{bc} &= D_{bs}-D_{cs} = v(T_b-T_c) = vt_{bc} \\ &= \sqrt{(x_s-x_b)^2+(y_s-y_b)^2+(z_s-0)^2} - \sqrt{(x_s-0)^2+(y_s-0)^2+(z_s-0)^2} \end{aligned} \qquad (3.8)$$

$$\begin{aligned} D_{dc} &= D_{ds}-D_{cs} = v(T_d-T_c) = vt_{dc} \\ &= \sqrt{(x_s-x_d)^2+(y_s-0)^2+(z_s-0)^2} - \sqrt{(x_s-0)^2+(y_s-0)^2+(z_s-0)^2} \end{aligned} \qquad (3.9)$$

where t_{ac}, t_{bc}, t_{dc} are the time differences of arrival of the sound wave between microphones $A\&C$, $B\&C$, $D\&C$ respectively, which we need to calculate.
Now from Eq. 3.3, Eq. 3.4 and Eq. 3.5,

$$t_{ac} = \frac{\sqrt{(x_s-0)^2+(y_s-y_a)^2+(z_s-0)^2} - \sqrt{(x_s-0)^2+(y_s-0)^2+(z_s-0)^2}}{v} \qquad (3.10)$$

$$t_{bc} = \frac{\sqrt{(x_s-x_b)^2+(y_s-y_b)^2+(z_s-0)^2} - \sqrt{(x_s-0)^2+(y_s-0)^2+(z_s-0)^2}}{v} \qquad (3.11)$$

$$t_{dc} = \frac{\sqrt{(x_s-x_d)^2+(y_s-0)^2+(z_s-0)^2} - \sqrt{(x_s-0)^2+(y_s-0)^2+(z_s-0)^2}}{v} \qquad (3.12)$$

Using Eq. 3.6, 3.7, 3.8 we can calculate time delay.

3.2 Estimating time delay from sensors

The source and the sensors are present in an anechoic space with the sensors lying in the far field of the source. The waves passing through the array are approximated as planar. The source is stationary w.r.t. sensor array.

3.2.1 Signal model

Let $x_1(n)$ and $x_2(n)$ be the samples received by two spatially separated sensors at an instant nT_s, where T_s is the inverse of sampling rate, F_s.

$$x_1(n) = \alpha_1 s(n) + n_1(n) \tag{3.13}$$
$$x_2(n) = \alpha_2 s(n-\tau) + n_2(n) \tag{3.14}$$

where $s(n)$ is the signal from source (undergoing WSS), $n_1(n)$, and $n_2(n)$ is the noise at each of the sensors which is uncorrelated with the signal, τ is the time delay, and α_1, α_2 are the attenuation parameters. The total length of each sequence x_1, x_2 is equal to N.

3.2.2 Normalized frequency based TDoA

This is a very crude approach to estimate time delay and was conceived by us during the initial phase of this project. This method works if and only if we have a single frequency in the signal. This is a very simple technique which uses the fact that source frequency is known.

Consider the continuous time signals $x_{a1}(t)$, $x_{a2}(t)$ received at the sensors. Assume no noise. Let F be the source frequency which is known. After sampling the CT signals we obtain DT sequences $x_1(n)$, $x_2(n)$. Consider samples within a window of size N. According to [12], the normalized frequency of the DT sequence is given by

$$f_0 = \frac{F}{F_s} \tag{3.15}$$

From the time-domain shifting property of DTFT:

$$\text{If, } x(n) \underset{DTFT}{\Longleftrightarrow} X(f)$$
$$\text{then, } x(n-D) \underset{DTFT}{\Longleftrightarrow} e^{-j2\pi f D} X(f)$$

it is evident that time shifting information is present in the phase of the signal.

Using the aforementioned points, the time difference estimate w.r.t. the reference signal x_1, $\hat{\tau}^{DS}$, is given by:

$$X_i = \sum_{n=0}^{N-1} x_i(n)e^{-j2\pi f_0 n},\ i = 1,2 \quad (3.16)$$

$$\hat{\tau}^{DS} = \frac{\angle X_2 - \angle X_1}{2\pi F} \quad (3.17)$$

Implementation Results

The following figure shows a plot between actual time difference and estimated difference when the source is moved around the sensor array by 360°, assuming no noise.

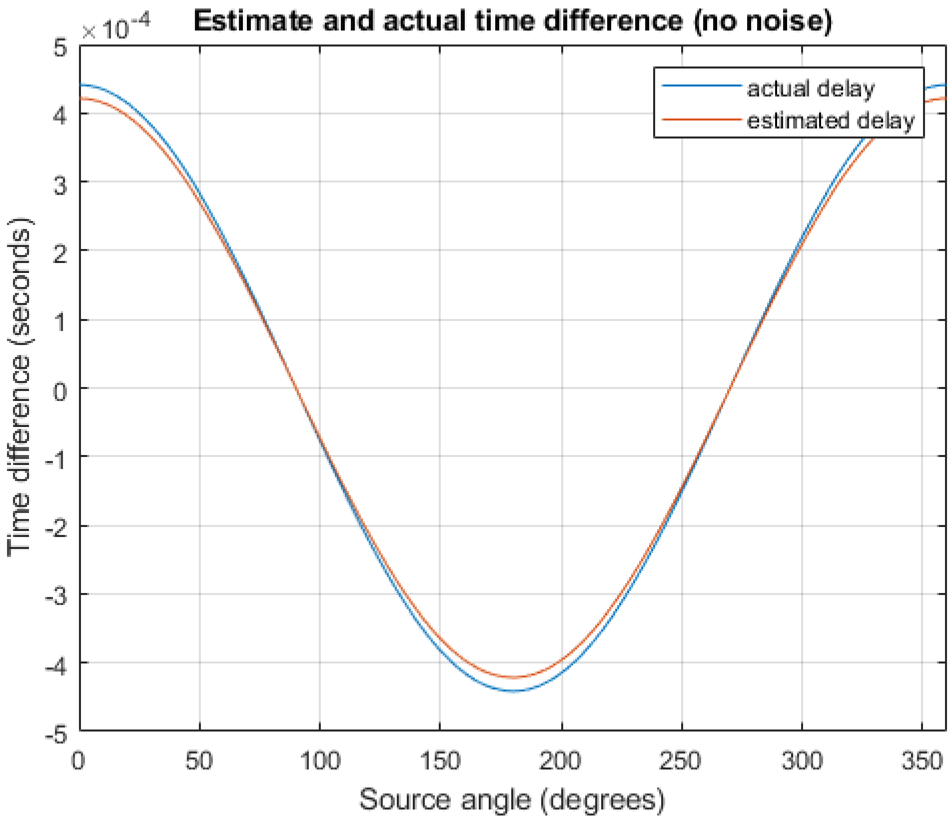


Figure 3.4: Actual and estimated time difference with no noise

When noise of varying levels is added and the actual and estimated time differences are plotted, we observe the following graphs:

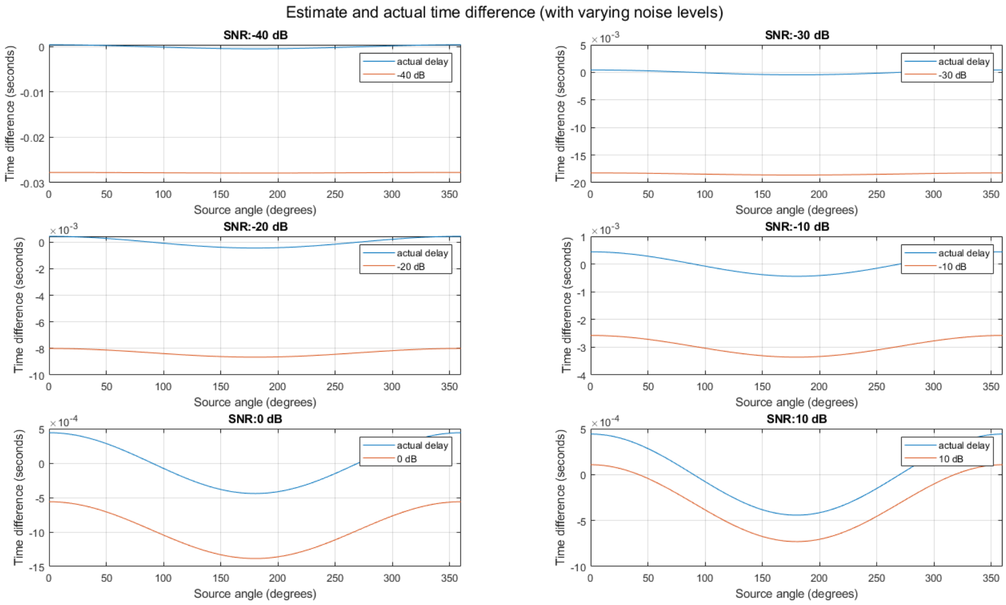


Figure 3.5: Actual and estimated time difference with varying noise - Normalized frequency method

Observations and Inferences

- To use this method with a noisy signal, a filtering step is mandatory.
- The error in TDoA estimate is maximum when source is located at angles 0°, 180°, 360°.
- The error in TDoA estimate even when no noise does not exactly overlap actual delay.

3.2.3 Cross-Correlation

Cross correlation function is an inner-product-type function that provides a measure of similarity between two waveforms [13]. Consider the aforementioned signal model - (3.3) and (3.4). The cross-correlation between the two observation signals is defined as (from section: 1.3):

$$\begin{aligned} R^{CC}_{x_1x_2}(p) &= E[x_1(n)x_2(n+p)] \\ &= \alpha_1\alpha_2 r^{CC}_{ss}(p-\tau) + \alpha_1 r^{CC}_{sn_2}(p+n) + \\ &\quad \alpha_2 r^{CC}_{sn_1}(p-n-\tau) + r^{CC}_{n_1n_2}(p) \end{aligned} \tag{3.18}$$

If we assume that $n_i(n)$ are uncorrelated with both the signal and the noise observed at the other sensor, it can be easily checked that $R^{CC}_{x_1x_2}(p)$ reaches maximum at $p = \tau$. Hence, the estimate of TDoA is given by

$$\hat{\tau}^{CC} = \arg\max_{p} R^{CC}_{x_1x_2}(p) \tag{3.19}$$

In the digital implementation of (3.9), only can estimate of cross-correlation is obtained, represented by $\hat{R}^{CC}_{x_1x_2}(p)$ [14].

The paper [4] discusses the practical implementation of cross-correlation function to estimate time difference. The cross-correlation method is a simple method to implement but for low SNR, ambiguity arises in detecting the peak. The cross-correlation method is part of a wide range of methods under GCC.

3.2.4 GCC-PHAT

About GCC

The generalized cross-correlation (GCC) algorithm was proposed by Knapp and Carter [5] and is the one of the most widely used algorithms. The method works when *priori* knowledge of the signal is available and performs moderately well in noisy and non-reverberant environments.

GCC Method

The TDoA between two microphones is obtained as lag time that maximizes cross correlation between the filtered signals of the microphone outputs:

$$\hat{\tau}^{GCC} = \arg\max_{\tau} R^{GCC}_{x_1x_2}(p) \tag{3.20}$$

where

$$R^{GCC}_{x_1x_2}(p) = \int_{-\infty}^{\infty} \nu(\omega) S_{x_1x_2}(\omega) e^{j\omega p} d\omega \tag{3.21}$$

is the GCC function,

$$S_{x_1x_2}(\omega) = X_1(\omega) X_2^*(\omega) \tag{3.22}$$

is cross-spectrum with

$$X_n(\omega) = \sum_n x_n(n) e^{-j\omega n},\ n = 1, 2 \tag{3.23}$$

and $\nu(\omega)$ is a frequency domain weighting function

There are many different choices for frequency-domain weighting function $\nu(\omega)$, leading to a variety of GCC methods[14].

Phase Transform Method

When we set the weighting function as:

$$\nu(\omega) = \frac{1}{|S_{x_1x_2}(\omega)|} \tag{3.24}$$

we get the phase transform (PHAT) method. Since all the TDoA information is present in the phase rather than amplitude, the amplitude is discarded from cross-spectrum.

Implementation Results

Two sensors are placed along `y - axis` and a source is moved around them, from 0 to 180°. For various levels of SNR, the estimation observed:

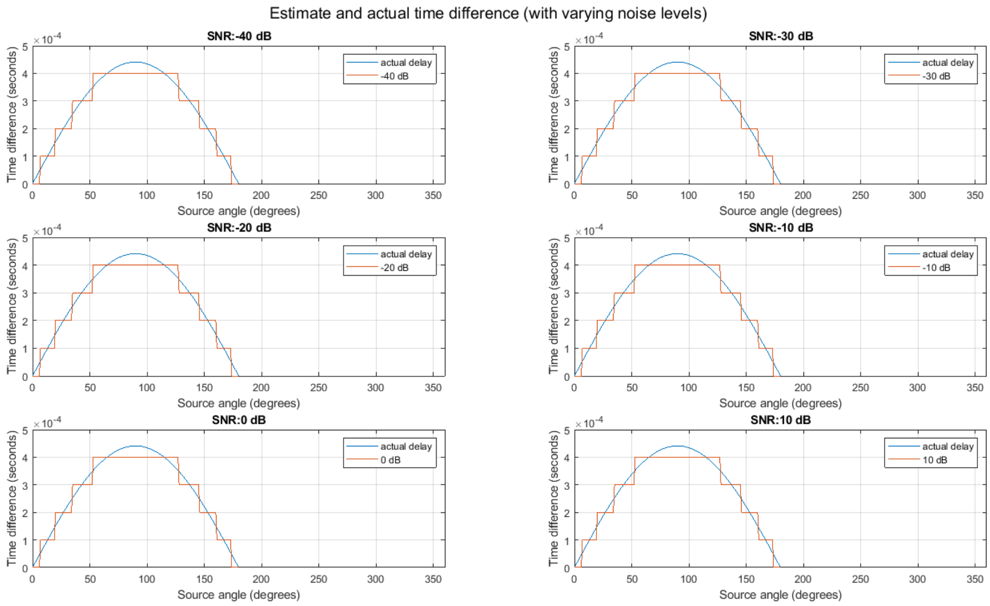


Figure 3.6: Actual and estimated time difference with varying noise - GC-CPHAT

The source is moved around the array in steps of 1°. A closer look into the values would provide a better understanding of the step behavior in estimation:

t_diff | tau_noise_est

1x181 double

	30	31	32	33	34	35	36	37	38	39	40
1	2.1389e-04	2.2059e-04	2.2722e-04	2.3379e-04	2.4028e-04	2.4670e-04	2.5305e-04	2.5932e-04	2.6551e-04	2.7162e-04	2.7764e-04

Figure 3.7: Actual TDoA for various locations of source

t_diff | tau_noise_est

6x181 double

	30	31	32	33	34	35	36	37	38	39	40
1	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04
2	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04
3	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04
4	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04
5	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04
6	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	2.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04	3.0000e-04

Figure 3.8: Estimated TDoA for various source location; each row shows values for ascending order of SNR levels.

Observations and Inferences

- The estimation plot is behaving in a step manner is because incorporation of a sub-sample delay as described in [15] has not been implemented. This method of TDoA would give relatively low error when we incorporate sub-sample delay estimation.

- This method is not opted because of an additional block to calculate sub-sample delay is required which increases computation cost.

3.2.5 Hilbert Transform based TDoA

The Hilbert Transform of a signal $r(t)$ is defined as [16]

$$\hat{r}(t) = \mathcal{H}\{r(t)\} = \frac{1}{\pi}\int_{-\infty}^{\infty} \frac{r(\tau)}{t-\tau} d\tau = \frac{1}{\pi t} \circledast r(t) \tag{3.25}$$

where the integral is a Cauchy Principal Value (CPV) and $\circledast$ denotes convolution.

The paper [10] derives an estimator for narrowband signal using the Hilbert transform. The derivation involves cross-correlation between the Hilbert transform of the reference signal and time delayed version of reference signal.

Let the energy of the narrowband signal be concentrated around $\pm f_0$. Let column vector of length N - $\hat{x}_1(n)$ - represent the Hilbert transform of the sampled version ($x_1(n)$) of the reference signal and column vector of same length, $x_2(n)$, represent the sampled version of the delayed signal. The time difference estimate, $\hat{\tau}^{HT}$, is given by

$$\hat{\tau}^{HT} = \frac{1}{2\pi f_0} \arcsin\left(-\frac{x_2^T \hat{x}_1}{x_1^T x_1}\right) \tag{3.26}$$

Implementation Results

The following figure shows a plot between actual time difference and estimated time difference when the source is moved around the sensor array from 0° to 360°, assuming no noise.

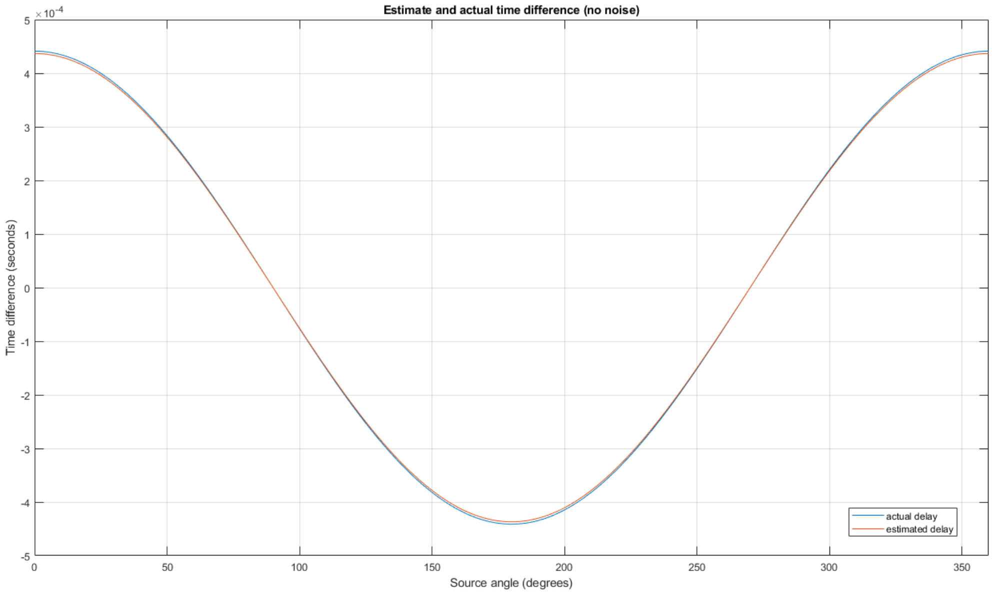


Figure 3.9: Actual and estimated time difference with no noise

When noise of varying levels is added and the actual and estimated time differences plotted, without the use of a pre-filter, the observations are:

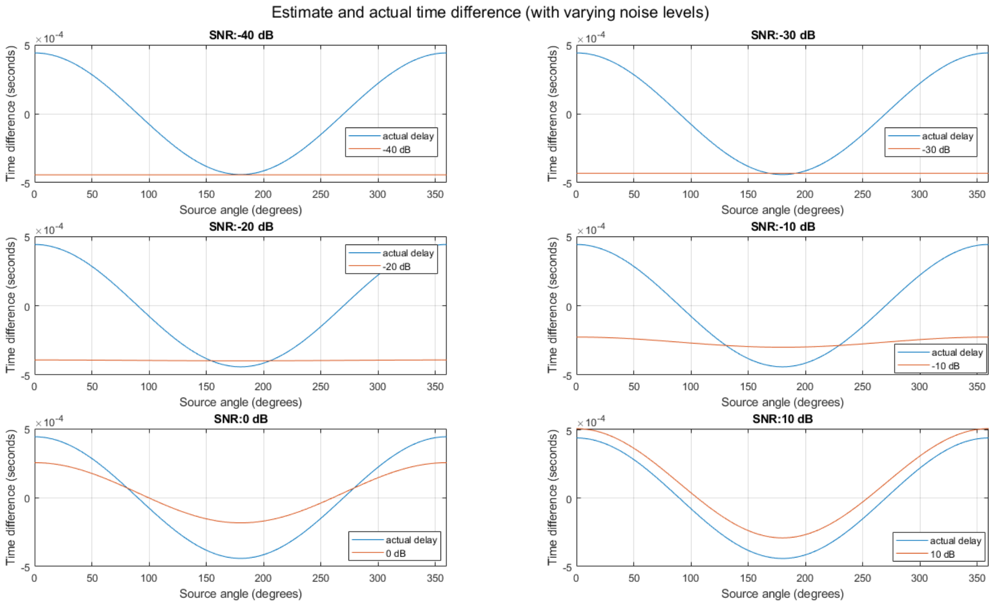


Figure 3.10: Actual and estimated time difference with varying noise - Hilbert transform based TDoA estimation

We have chosen to go ahead with this estimator because:

- The estimator has been specially derived for narrowband signals and we are dealing with narrowband signal in our project.
- The no noise graph is approximately same as the actual graph and it is difficult to distinguish both of them. Hence, by using a filter we can reduce the noise and use this estimator.
- The computation cost can be reduced by using FFT algorithm such that overall time complexity of the estimator is approximately $\mathcal{O}(N\,log\,N)$.
- On comparing the plot in figure 3.4 and figure 3.10, we can see that in the ideal case, the Hilbert transform method gives better results than the method we proposed. Since, the localization algorithms require higher level of precision, we go ahead with this method.

We now present the derivation of the mathematical formulae which make use of TDoA estimates to localize a source in 2D *(right triangle configuration)* and 3D *(square configuration)* space.

3.3 Right triangle configuration

We aim to obtain the azimuth angle of arrival(AoA) of the source signal w.r.t `x-axis` at origin.

Once the time difference between microphone at point P and R, t_{pr} and between Q and R, t_{qr} are known using the TDoA estimation methods discussed in the previous chapter, equation (3.1) and (3.2) can be used to obtain two estimates of the angle of arrival. The final estimate of angle of arrival is given by

$$\hat{\theta} = \frac{\hat{\theta}_{pr} + \hat{\theta}_{qr}}{2} \tag{3.27}$$

where $\hat{\theta}_{pr}$, $\hat{\theta}_{qr}$ represent estimates of angle of arrival obtained from equation (3.1) and (3.2) respectively.

Implementation Results

The following plot is obtained when a source is moved around the sensors from 0° to 360°, without noise:

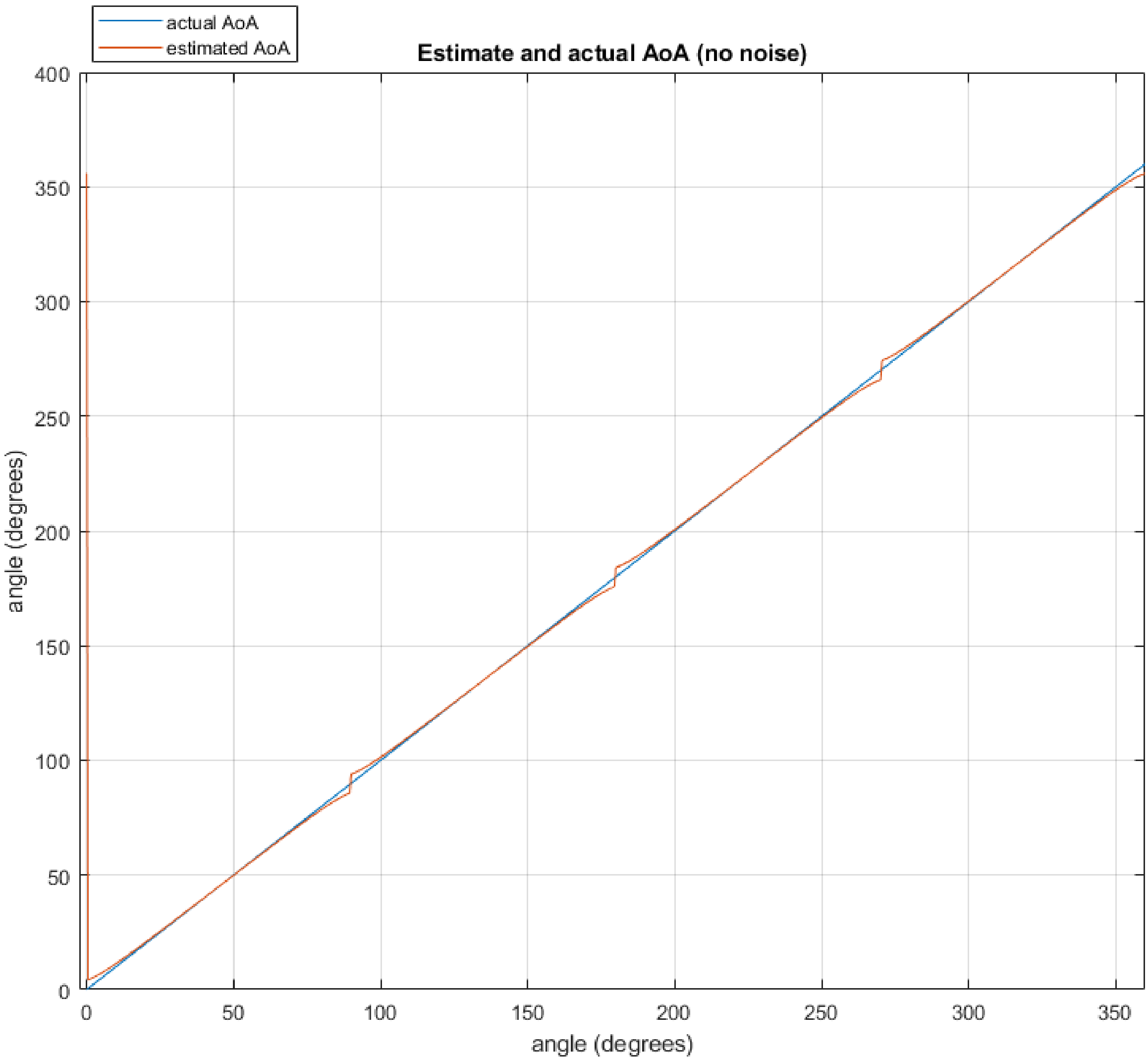


Figure 3.11: Actual and estimated angle of arrival, without noise

Observations

- When the source is located at $0°$, $\hat{\theta} = 356°$. Considering $0°$ and $360°$ equivalent, the error is around $4°$.
- $\hat{\theta}$ deviates a more than $|2°|$ when it is in the range $\gamma - 10° \leq \hat{\theta} \leq \gamma + 10°$, where $\gamma = 90°, 180°, 270°, 360°$. The maximum error in this range is at approximately $\gamma \pm 5°$, and is of the value $\pm 4°$.

- As per the assumptions, since the source is very far away from the sensor, an error of $\pm 4°$ is acceptable for the ideal case. In the next case of signals with noise, we aim to keep the error at max less than $|10°|$.

The following plot shows the estimated and actual angle of arrival when a noisy signal with -10 dB, 15 dB, 20 dB, and 30 dB SNR is emanated from the source. The custom filter described in *Appendix B* has been used to eliminate noise:

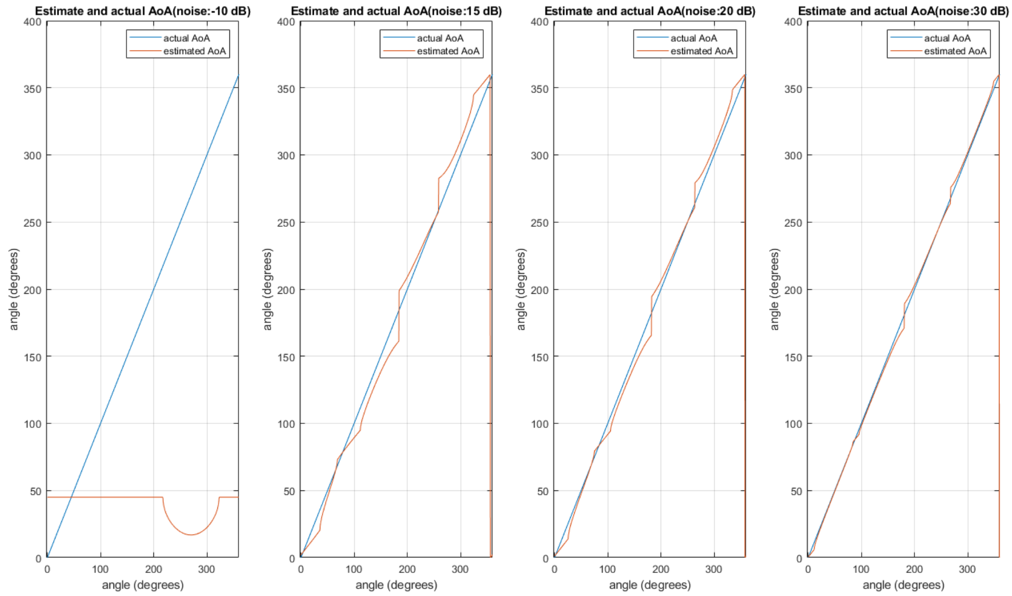


Figure 3.12: Actual and estimated angle of arrival, with noise

Observations

- When the SNR is 20 dB, the maximum error in the AoA estimation is $\pm 10°$.
- When the SNR is less than 15 dB, the graphs of estimated and actual AoA are no where close to each other.
- The maximum error when SNR is 15 dB is around 20°.

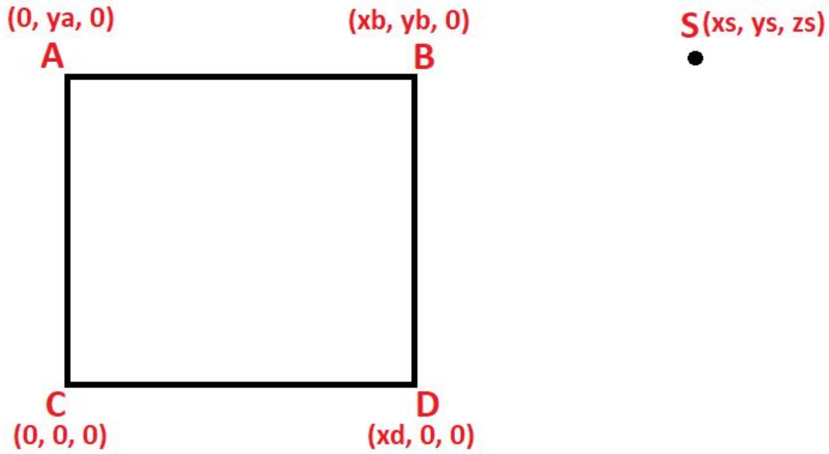
To improve the estimation, work needs to be done in designing a filter to eliminate the noise such that SNR is atleast close to 20 dB. At a hardware level, this can be achieved in two stages:

1. Analog noise filtering
2. Digital noise filtering

Figure (4.1) provides the motivation to improve on the noise eliminating pre-filter.

3.4 Square configuration

We aim to obtain the 3-D coordinates of the source using the time differences of arrival.



As defined in section 3.1.2, t_{ac}, t_{bc}, t_{dc} are the time differences of arrival of the sound wave between microphones $A\&C$, $B\&C$, $D\&C$ respectively
We also define,

$$\begin{aligned} t_{ba} &= T_b - T_a \\ &= T_b - T_a + T_c - T_c \\ &= (T_b - T_c) - (T_a - T_c) \\ &= t_{bc} - t_{ac} \end{aligned} \tag{3.28}$$

$$
\begin{aligned}
t_{bd} &= T_b - T_d \\
&= T_b - T_d + T_c - T_c \\
&= (T_b - T_c) - (T_d - T_c) \\
&= t_{bc} - t_{dc} \qquad (3.29)
\end{aligned}
$$

Now recall Eq. 3.3, 3.4, 3.5, 3.6 and using Eq. 4.2, 4.3 we define the following,
And since this is a square configuration, $y_a = x_b = y_b = x_d = d$, where d is a positive real number.[3]

$$
\begin{aligned}
D_{ba} &= D_{bs} - D_{as} = v(T_b - T_a) = vt_{ba} = v(t_{bc} - t_{ac}) \\
&= \sqrt{(x_s - d)^2 + (y_s - d)^2 + (z_s - 0)^2} - \sqrt{(x_s - 0)^2 + (y_s - d)^2 + (z_s - 0)^2} \qquad (3.30) \\
D_{bd} &= D_{bs} - D_{ds} = v(T_b - T_d) = vt_{bd} = v(t_{bc} - t_{dc}) \\
&= \sqrt{(x_s - d)^2 + (y_s - d)^2 + (z_s - 0)^2} - \sqrt{(x_s - d)^2 + (y_s - 0)^2 + (z_s - 0)^2} \qquad (3.31) \\
D_{ac} &= D_{as} - D_{cs} = v(T_a - T_c) = vt_{ac} \\
&= \sqrt{(x_s - 0)^2 + (y_s - d)^2 + (z_s - 0)^2} - \sqrt{(x_s - 0)^2 + (y_s - 0)^2 + (z_s - 0)^2} \qquad (3.32) \\
D_{dc} &= D_{ds} - D_{cs} = v(T_d - T_c) = vt_{dc} \\
&= \sqrt{(x_s - d)^2 + (y_s - 0)^2 + (z_s - 0)^2} - \sqrt{(x_s - 0)^2 + (y_s - 0)^2 + (z_s - 0)^2} \qquad (3.33)
\end{aligned}
$$

Now taking Eq. 4.4,

$$
\sqrt{(x_s - 0)^2 + (y_s - d)^2 + (z_s - 0)^2} = \sqrt{(x_s - d)^2 + (y_s - d)^2 + (z_s - 0)^2} - D_{ba}
$$

Squaring both the sides,

$$
\begin{aligned}
x_s^2 + y_s^2 + d^2 - 2dy_s + z_s^2 &= x_s^2 + d^2 - 2dx_s + y_s^2 + d^2 - 2dy_s + z_s^2 + D_{ba}^2 \\
&\quad -2D_{ba}\sqrt{(x_s - d)^2 + (y_s - d)^2 + (z_s - 0)^2}
\end{aligned}
$$

[3] d is in accordance with equation 2.2

$$\begin{aligned}
\implies 0 &= -2dx_s + d^2 + D_{ba}^2 \\
&\quad -2D_{ba}\sqrt{(x_s-d)^2+(y_s-d)^2+(z_s-0)^2} \\
2D_{ba}\sqrt{(x_s-d)^2+(y_s-d)^2+(z_s-0)^2} &= D_{ba}^2 - 2dx_s + d^2 \\
\sqrt{(x_s-d)^2+(y_s-d)^2+(z_s-0)^2} &= \frac{D_{ba}^2 - 2dx_s + d^2}{2D_{ba}}
\end{aligned} \tag{3.34}$$

Now taking Eq. 4.5,

$$\sqrt{(x_s-d)^2+(y_s-0)^2+(z_s-0)^2} = \sqrt{(x_s-d)^2+(y_s-d)^2+(z_s-0)^2} - D_{bd}$$

Squaring both the sides,

$$\begin{aligned}
x_s^2 + d^2 - 2dx_s + y_s^2 + z_s^2 &= x_s^2 + d^2 - 2dy_s + y_s^2 + d^2 - 2dy_s + z_s^2 + D_{bd}^2 \\
&\quad -2D_{bd}\sqrt{(x_s-d)^2+(y_s-d)^2+(z_s-0)^2} \\
\implies 0 &= -2dx_s + d^2 + D_{bd}^2 \\
&\quad -2D_{bd}\sqrt{(x_s-d)^2+(y_s-d)^2+(z_s-0)^2} \\
2D_{bd}\sqrt{(x_s-d)^2+(y_s-d)^2+(z_s-0)^2} &= D_{bd}^2 - 2dy_s + d^2 \\
\sqrt{(x_s-d)^2+(y_s-d)^2+(z_s-0)^2} &= \frac{D_{bd}^2 - 2dy_s + d^2}{2D_{bd}}
\end{aligned} \tag{3.35}$$

Now from Eq. 4.8 and Eq. 4.9,

$$\begin{aligned}
\frac{D_{ba}^2 - 2dx_s + d^2}{2D_{ba}} &= \frac{D_{bd}^2 - 2dy_s + d^2}{2D_{bd}} \\
\implies (2dD_{ba})y - (2dD_{bd})x &= (D_{ba} - D_{bd})(d^2 - D_{ba}D_{bd})
\end{aligned} \tag{3.36}$$

Now taking Eq. 4.6,

$$\sqrt{(x_s-0)^2+(y_s-d)^2+(z_s-0)^2} = \sqrt{(x_s-0)^2+(y_s-0)^2+(z_s-0)^2}+D_{ac}$$

Squaring both the sides,

$$\begin{aligned}
x_s^2+y_s^2+d^2-2dy_s+z_s^2 &= x_s^2+y_s^2+z_s^2+D_{ac}^2+2D_{ac}\sqrt{x_s^2+y_s^2+z_s^2} \\
\implies d^2-2dy_s &= D_{ac}^2+2D_{ac}\sqrt{x_s^2+y_s^2+z_s^2} \\
2D_{ac}\sqrt{x_s^2+y_s^2+z_s^2} &= d^2-2dy_s-D_{ac}^2 \\
\sqrt{x_s^2+y_s^2+z_s^2} &= \frac{d^2-2dy_s-D_{ac}^2}{2D_{ac}}
\end{aligned} \tag{3.37}$$

Now taking Eq. 4.7,

$$\sqrt{(x_s-d)^2+(y_s-0)^2+(z_s-0)^2} = \sqrt{(x_s-0)^2+(y_s-0)^2+(z_s-0)^2}+D_{dc}$$

Squaring both the sides,

$$\begin{aligned}
x_s^2+d^2-2dx_s+y_s^2+z_s^2 &= x_s^2+y_s^2+z_s^2+D_{dc}^2+2D_{dc}\sqrt{x_s^2+y_s^2+z_s^2} \\
\implies d^2-2dx_s &= D_{dc}^2+2D_{dc}\sqrt{x_s^2+y_s^2+z_s^2} \\
2D_{dc}\sqrt{x_s^2+y_s^2+z_s^2} &= d^2-2dx_s-D_{dc}^2 \\
\sqrt{x_s^2+y_s^2+z_s^2} &= \frac{d^2-2dx_s-D_{dc}^2}{2D_{dc}}
\end{aligned} \tag{3.38}$$

Now from Eq. 4.11 and Eq. 4.12,

$$\begin{aligned}
\frac{d^2-2dy_s-D_{ac}^2}{2D_{ac}} &= \frac{d^2-2dx_s-D_{dc}^2}{2D_{dc}} \\
\implies (2dD_{dc})y-(2dD_{ac})x &= (D_{dc}-D_{ac})(d^2+D_{dc}D_{ac})
\end{aligned} \tag{3.39}$$

Eq. 4.10 and 4.13 are linear equations with variables as x,y. Solving them further gives x and y, which upon substitution in Eq.4.12 gives $\pm z$.

$$x = \frac{D_{ba}(D_{dc} - D_{ac})(d^2 + D_{dc}D_{ac}) - D_{dc}(D_{ba} - D_{bd})(d^2 + D_{ba}D_{bd})}{2d(D_{bd}D_{dc} - D_{ac}D_{ba})} \tag{3.40}$$

Substituting x in 4.10

$$y = \frac{(D_{dc} - D_{ac})(d^2 + D_{dc}D_{ac}) + 2dD_{ac}}{2dD_{dc}} \tag{3.41}$$

$$z = \sqrt{(\frac{d^2 - 2dx_s - D_{dc}^2}{2D_{dc}})^2 - x^2 - y^2} \tag{3.42}$$

Hence, using equations (3.40), (3.41), and (3.42), one can find the location of the source.

Chapter 4

Work Done

Team Member: Mayank N. Mehta [EDM18B037]
Contributed in:

- Conceptualizing Normalized frequency method for TDoA estimation
- Deriving equations for right triangle configuration for AoA estimation
- Conceptualizing and building the filter
- Writing MATLAB code to implement Normalized frequency method for TDoA estimation and to observe the results
- Incorporating environmental parameters into the final simulation
- Drafting project's initial presentation, progress report, and final report

Team Member: Vishva Bhate [EDM18B054]
Contributed in:

- Conceptualizing Normalized frequency method for TDoA estimation
- Deriving equations for square configuration for source coordinate estimation
- Writing MATLAB code to implement GCC-PHAT and Hilbert transform methods for TDoA estimation and to observe the results
- Implementing complete simulation of the project in MATLAB
- Drafting project's initial presentation, progress report, and final report

Chapter 5

Block Diagram

The block diagram in figure (5.1) shows various blocks involved in the final simulation tool created as the output of this report.

- The terms written outside a rectangular box represent input and the blocks themselves represent the process.
- **Block 1** generates the attenuation of the source signal in a terrestrial environment.
- **Block 2** aims to simulate the real world signal as sampled by an ADC.
- **Block 3** filters out noise present in the sampled signal. This project does not focus much on this block as the type of filter to be implemented depends on the type of environment and acceptable error range for an application. However, we have presented a design constraint on the output of this block for proper results.
- **Block 4** estimates the time difference of arrival (TDoA) between the sensors.
- **Block 5** implements the localization algorithm.

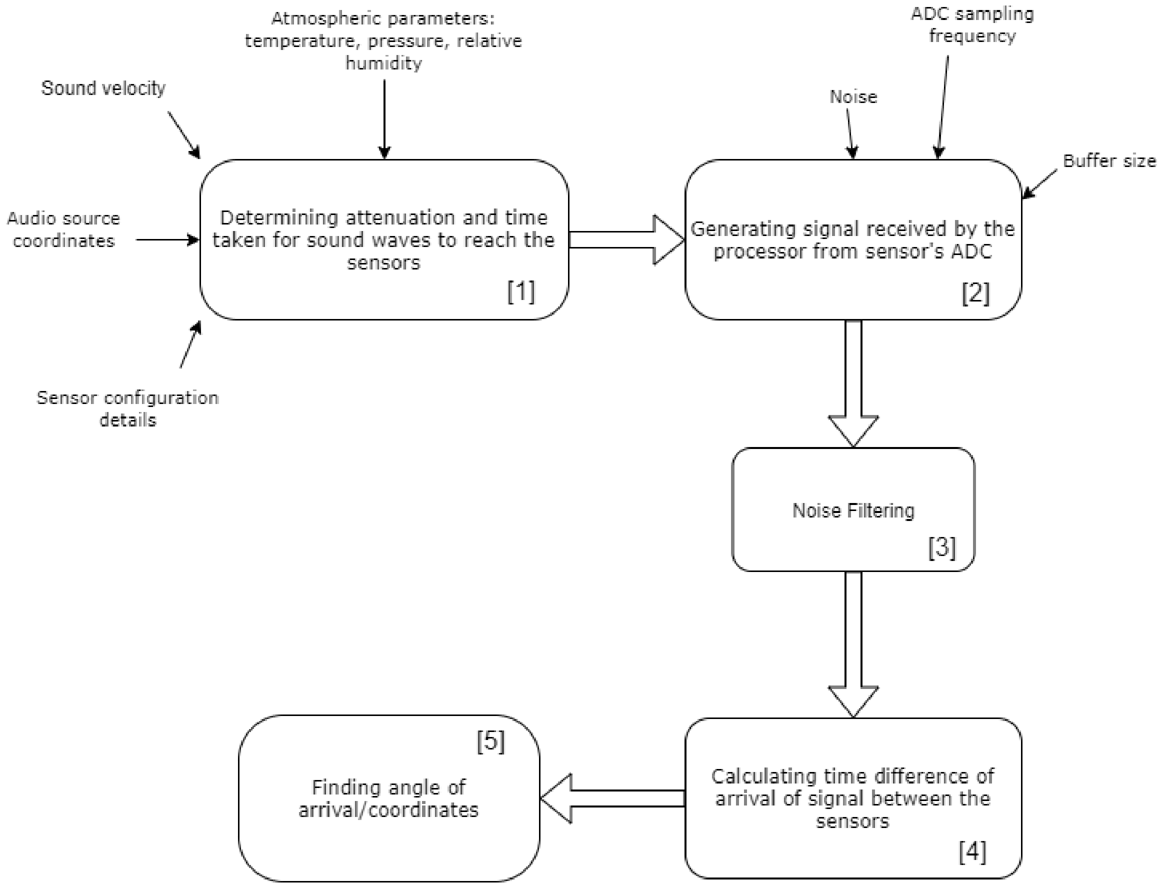


Figure 5.1: Block diagram of the simulation tool

Chapter 6

Execution Procedure

This chapter discusses about

- files that make up the simulation tool
- how to run the simulation

6.1 Simulation Files

S.No.	File Name	About
1	ASL_SimTool.m	**MATLAB script file** | interacts with the user and takes the environmental, sensor and source configurations. All the other files in the tool kit are accessed through this script.
2	get_ActualTimeDiff.m	**MATLAB function file** | returns actual time difference between the signals received by various sensors based on source and sensor configuration.
3	get_AmpAtten.m	**MATLAB function file** | returns attenuation parameters due to environmental effects.
4	get_AoA.m	**MATLAB function file** | returns estimated angle of arrival of the source signal.
5	get_SoundSpeed.m	**MATLAB function file** | returns speed of sound due to various environmental parameters.
6	get_TDoAEstimate.m	**MATLAB function file** | returns the estimate of time difference of arrival between sensors.
7	get_xyz.m	**MATLAB function file** | returns the coordinates of the source signal.
8	single_freq_filter.m	**MATLAB function file** | returns filtered signal.

6.2 How to execute

To run the simulation and test the estimation algorithm,

1. Run the MATLAB script `ASL_SimTool.m`

2. The program takes the environmental parameters, sensor configuration, and signal source configuration as the input with prompts appearing sequentially.

3. Observe the output for the given configuration and design the hardware for *acoustic source localization*.

Chapter 7

MATLAB Implementation

This chapter contains all the listings of the simulation toolkit developed as a part of this project.

ASL_SimTool.m

```
%% Acoustic Source Localization Simulation Tool
% -------------------------------------------------------------
% A modular simulation toolkit for testing acoustic source localization
% in terrestrial environment. This toolkit is the product of the project
% titled *Acoustic Source Localization Techniques*.
% -------------------------------------------------------------
%
% Creators: Vishva Nilesh Bhate and Mayank Navneet Mehta
% ..............................................................

%% Clearing workspace

close all
clear
clc

fprintf('Welcome to ASL-SimTool\n\n');

%% Environmental configuration

disp('---ENVIRONMENTAL CONFIGURATION---');

fprintf("\nDescribe the terrestrial environment");

```

```
T = input('\nRoom temperature (in degree C): ');
if isempty(T)
    disp('Invalid temperature');
    return;
end

P = input('Absolute pressure (in Pascals | at ...
    sea-level:101.325e5) : ');
if isempty(P)
    disp('Invalid pressure');
    return;
end

RH = input('Relative humidity in between [0,1]: ');
if isempty(RH) || (RH > 1 || RH < 0)
    disp('Invalid relative humidity');
    return;
end

SNR = input('Noise level (in dB): ');
if isempty(SNR)
    disp('Invalid signal to noise ration');
    return;
end

%% Sensor configuration

fprintf('\n---SENSOR CONFIGURATION---');

fprintf("\nSelect your sensor configuration:\n[1] Right ...
    triangle\n[2] Square");

sensor_config = input('\nYour option: ');

if ¬isempty(sensor_config)
    if ¬(isequal(sensor_config,1) || isequal(sensor_config,2))
        disp('Invalid configuration. Enter number 1 or 2 ...
            only.');
        return;
    end
else
    disp('Invalid configuration');
    return;
end

if (sensor_config == 1)
    sens_coord = zeros(2,1);    % sensor coordinate vector
    fprintf('Sensor R is located at the origin (0,0)');
```

```
    sens_coord(2) = input('\nEnter the x-coordinate of Q ...
        (in m): ');
    sens_coord(1) = input('Enter the y-coordinate of P (in ...
        m): ');
elseif (sensor_config == 2)
    sens_coord = zeros(4,3);    % sensor coordinate vector
    fprintf('Sensor C is located at the origin (0,0)');
    fprintf('\nSince it is a square configuration distance ...
        between adjacent sensors is same');
    fprintf('\n*A(0,d,0)    *B(d,d,0)');
    fprintf('\n                       ');
    fprintf('\n*C(0,0,0)    *D(d,0,0)');
    size_square = input('\nEnter distance between any two ...
        sensors, d(in m): ');
    sens_coord(1,:) = [0 size_square 0];
    sens_coord(2,:) = [size_square size_square 0];
    sens_coord(3,:) = [0 0 0];
    sens_coord(4,:) = [size_square 0 0];
end

Fs = input('Enter the sampling frequency (in Hz): ');
if isempty(Fs)
    disp('Invalid sampling frequency');
    return;
end

N = input('Enter the buffer size (preferable power of 2): ');
if isempty(N)
    disp('Invalid buffer size');
    return;
end

%% Source configuration

fprintf('\n---SOURCE CONFIGURATION---');

F = input('\nEnter the source frequency: ');
if isempty(F)
    disp('Invalid source frequency');
    return;
elseif ¬(F < Fs/2)
    disp('WARNING: Nyquist theorem in time domain is not ...
        statisfied.');
end

A = input('Enter the source amplitude: ');
if isempty(A)
    disp('Invalid source amplitude');
    return;
```

```
end

src_function = input('Source function:\n[1] Sine\n[2] ...
    Cosine\nYour option: ');
if ¬(src_function == 1 || src_function == 2)
    disp('Invalid source function');
end

if (sensor_config == 1)
    src_coord = zeros(2,1);     % source coordinate vector
    src_coord(1) = input('Enter the x-coordinate of S (in ...
        m): ');
    src_coord(2) = input('Enter the y-coordinate of S (in ...
        m): ');
elseif (sensor_config == 2)
    src_coord = zeros(3,1);     % source coordinate vector
    src_coord(1) = input('Enter the x-coordinate of S (in ...
        m): ');
    src_coord(2) = input('Enter the y-coordinate of S (in ...
        m): ');
    src_coord(3) = input('Enter the z-coordinate of S (in ...
        m): ');
end

[v,alpha] = get_SoundSpeed(T, P, RH, F);
lambda = v/F;                          % source wavelength
% Validating far-field condition and spatial sampling theorem
if (sensor_config == 1)
    if ¬((sens_coord(1) ≤ lambda/2) && (sens_coord(2) ≤ ...
        lambda/2) )
        disp('WARNING: Nyquist theorem in spatial domain ...
            is not statisfied.');
    end

    if ¬((sqrt(src_coord(1)ˆ2 + src_coord(2)ˆ2) > ...
        (2*(sens_coord(1)ˆ2)/lambda))...
  &&(sqrt(src_coord(1)ˆ2 + src_coord(2)ˆ2) > ...
      (2*(sens_coord(2)ˆ2)/lambda)))

        disp('Please change the parameters to satisfy the ...
            far field condition');
        return;
    end

elseif (sensor_config == 2)
    if ¬(size_square ≤ lambda/2)
        disp('WARNING: Nyquist theorem in spatial domain ...
            is not statisfied.');
    end
```

```
    if ¬((sqrt(src_coord(1)^2 + src_coord(2)^2 + ...
        src_coord(3)^2)) > (2*(sens_coord(1)^2)/lambda))...
  &&((sqrt(src_coord(1)^2 + src_coord(2)^2 + ...
      src_coord(3)^2)) > (2*(sens_coord(2)^2)/lambda))...
  &&((sqrt(src_coord(1)^2 + src_coord(2)^2 + ...
      src_coord(3)^2)) > (2*(sens_coord(3)^2)/lambda))
        disp('Please change the parameters to satisfy the ...
            far field condition');
        return;
    end
end

clear lambda;        % clearing lambda variable as it is of ...
    no use

%% THE SIMULATION

fprintf('\n---SIMULATING---');

% Time instances of sampling; length(n) = N
n = 0:1/Fs:(N-1)/Fs;

% Generating noise
if (sensor_config == 1)
    noise = zeros(3,length(n));
    noise(1,:) = randn(size(n));
    noise(2,:) = randn(size(n));
    noise(3,:) = randn(size(n));
elseif (sensor_config == 2)
    noise = zeros(4,length(n));
    noise(1,:) = randn(size(n));
    noise(2,:) = randn(size(n));
    noise(3,:) = randn(size(n));
    noise(4,:) = randn(size(n));
end


if (sensor_config == 1)
    % Get attenuation parameters
    [amp_c, amp_a, amp_b] = get_AmpAtten(A, alpha, ...
        src_coord,...
                                [0 ...
                                    sens_coord(2);sens_coord(1) ...
                                    0]);
    % Get actual time delays
    [t_ac, t_bc] = get_ActualTimeDiff(src_coord, ...
        sens_coord, v);

```

```
    fprintf('\nActual time delay:\nt_pr = %f \nt_qr = %f', ...
        t_ac, t_bc);

    % Generating signal
    if(src_function == 1)
        % Sine
        x_c = amp_c*sin(2*pi*F*n);
        x_a = amp_a*sin(2*pi*F*(n + t_ac));
        x_b = amp_b*sin(2*pi*F*(n + t_bc));
    elseif (src_function == 2)
        % Cosine
        x_c = amp_c*cos(2*pi*F*n);
        x_a = amp_a*cos(2*pi*F*(n + t_ac));
        x_b = amp_b*cos(2*pi*F*(n + t_bc));
    end

    % Adding noise
    x_c = x_c + ((norm(x_c)/norm(noise(1,:))) * ...
        10^(-SNR/20))*noise(1,:);
    x_a = x_a + ((norm(x_a)/norm(noise(2,:))) * ...
        10^(-SNR/20))*noise(2,:);
    x_b = x_b + ((norm(x_b)/norm(noise(3,:))) * ...
        10^(-SNR/20))*noise(3,:);

    % Filtering
    x_a_filt = single_freq_filter(x_c);
    x_b_filt = single_freq_filter(x_a);
    x_c_filt = single_freq_filter(x_b);

    % TDoA Estimation
    [tau_est_p, tau_est_q] = get_TDoAEstimate(x_a_filt, F, ...
        x_b_filt, x_c_filt);

    fprintf('\nEstimate time delay:\ntau_est_p = %f ...
        \ntau_est_q = %f',...
            tau_est_p, tau_est_q);

    % AoA Estimation
    theta_est = ...
        get_AoA(sens_coord(1),sens_coord(2),tau_est_p, ...
        tau_est_q,v);

    fprintf('\nActual angle: %f\nEstimated angle: %f',...
            rad2deg(atan2(src_coord(2),src_coord(1))),theta_est);

elseif (sensor_config == 2)
    % Get attenuation parameters
    [amp_c, amp_a, amp_b, amp_d] = get_AmpAtten(A, alpha, ...
        src_coord,...
```

```
        [sens_coord(3,1) sens_coord(1,1) sens_coord(2,1) ...
            sens_coord(4,1);sens_coord(3,2) ...
            sens_coord(1,2) sens_coord(2,2) ...
            sens_coord(4,2);sens_coord(3,3) ...
            sens_coord(1,3) sens_coord(2,3) sens_coord(4,3)]);
    % Get actual time delays
    [t_ac, t_bc, t_dc] = get_ActualTimeDiff(src_coord, ...
        sens_coord, v);

    fprintf('\nActual time delay:\nt_ac = %f \nt_bc = %f ...
        \nt_dc = %f', t_ac, t_bc, t_dc);

    % Generating signal
    if(src_function == 1)
        % Sine
        x_c = amp_c*sin(2*pi*F*n);
        x_a = amp_a*sin(2*pi*F*(n + t_ac));
        x_b = amp_b*sin(2*pi*F*(n + t_bc));
        x_d = amp_d*sin(2*pi*F*(n + t_dc));
    elseif (src_function == 2)
        % Cosine
        x_c = amp_c*cos(2*pi*F*n);
        x_a = amp_a*cos(2*pi*F*(n + t_ac));
        x_b = amp_b*cos(2*pi*F*(n + t_bc));
        x_d = amp_d*cos(2*pi*F*(n + t_dc));
    end

    % Adding noise
    x_c = x_c + ((norm(x_c)/norm(noise(1,:))) * ...
        10^(-SNR/20))*noise(1,:);
    x_a = x_a + ((norm(x_a)/norm(noise(2,:))) * ...
        10^(-SNR/20))*noise(2,:);
    x_b = x_b + ((norm(x_b)/norm(noise(3,:))) * ...
        10^(-SNR/20))*noise(3,:);
    x_d = x_d + ((norm(x_d)/norm(noise(4,:))) * ...
        10^(-SNR/20))*noise(4,:);

    % Filtering
    x_a_filt = single_freq_filter(x_a);
    x_b_filt = single_freq_filter(x_b);
    x_c_filt = single_freq_filter(x_c);
    x_d_filt = single_freq_filter(x_d);

    %TDoA Estimation
    [tau_est_ac, tau_est_bc, tau_est_dc] = ...
        get_TDoAEstimate(x_c_filt, F, x_a_filt, x_b_filt, ...
        x_d_filt);

```

```
    fprintf('\nEstimate time delay:\ntau_est_ac = %f ...
       \ntau_est_bc = %f \ntau_est_dc = %f',...
            tau_est_ac, tau_est_bc, tau_est_dc);

    % Source coordinates estimation
    [x,y,z] = get_xyz(tau_est_ac, tau_est_bc, ...
       tau_est_dc,v,size_square);

    fprintf('\nx: %f \ny: %f \nz: %f',...
            x,y,z);
end
```

get_ActualTimeDiff.m

```
function [td_1, td_2, varargout] = get_ActualTimeDiff(s, ...
    rx, v)
%%
% Function to obtain actual time delay
% Input parameters ...
   -------------------------------------------------------------
%
% s            : [2x1] or [3x1] vector of source coordinates
% rx           : [2x1] vector consisting of distance ...
   between origin and receiver
%                (in m) or [4x3] vector of receiver coordinates
% v            : speed of sound (in m/s)
%
% Output parameters ...
   ------------------------------------------------------------
%
% td_1      : TDoA between reference and node 1
% td_2      : TDoA between reference and node 2
% td_3      : TDoA between reference and node 3
%-----------------------------------------------------------------
 nargoutchk(2,3);

 if(nargout == 2)
     src_ang = atan2(s(2),s(1));
     % equation (3.2)
     td_1 = rx(1)*sin(src_ang)/v;
     td_2 = rx(2)*cos(src_ang)/v;
 elseif (nargout == 3)
    td_1 = (1/v)*(sqrt((s(1) - rx(1,1))^2 + (s(2) - ...
       rx(1,2))^2 + (s(3) - rx(1,3))^2) - ...
       sqrt((s(1)-rx(3,1))^2 + (s(2)-rx(3,2))^2 + ...
       (s(3)-rx(3,3))^2));
```

```
    td_2 = (1/v)*(sqrt((s(1) - rx(2,1))^2 + (s(2) - ...
        rx(2,2))^2 + (s(3) - rx(2,3))^2) - ...
        sqrt((s(1)-rx(3,1))^2 + (s(2)-rx(3,2))^2 + ...
        (s(3)-rx(3,3))^2));
    varargout{1} = (1/v)*(sqrt((s(1) - rx(4,1))^2 + (s(2) ...
        - rx(4,2))^2 + (s(3) - rx(4,3))^2) - ...
        sqrt((s(1)-rx(3,1))^2 + (s(2)-rx(3,2))^2 + ...
        (s(3)-rx(3,3))^2));
 end

end
```

get_AmpAtten.m

```
function [amp_ref, amp_1, amp_2, varargout] = ...
    get_AmpAtten(A, alpha, s, rx)
%%
% Function to obtain environmental attenuation parameters
% Input parameters ...
    ------------------------------------------------------------
%
% A             : amplitude of source
% s             : [2x1] or [3x1] vector of source coordinates
% rx            : [2x2] or [3x2] vector of receiver coordinates.
%
% Output parameters ...
    -----------------------------------------------------------
%
% amp_ref       : amplitude of signal received at reference node
% amp_1         : amplitude of signal received at node 1
% amp_2         : amplitude of signal received at node 2
% amp_3         : amplitude of signal received at node 3
%--------------------------------------------------------------
 nargoutchk(3,4);

 if(nargout == 3)
     amp_ref = A*exp(-alpha*sqrt( s(1)^2  + s(2)^2 ));
     amp_1 = A*exp(-alpha*sqrt( (s(1) - rx(1,1))^2 + (s(2) ...
         - rx(2,1))^2 ));
     amp_2 = A*exp(-alpha*sqrt( (s(1) - rx(1,2))^2 + (s(2) ...
         - rx(2,2))^2 ));
 elseif (nargout == 4)
    amp_ref = A*exp(-alpha*sqrt( (s(1) - rx(1,1))^2 + ...
        (s(2) - rx(2,1))^2 + (s(3) - rx(3,1))^2 ));
    amp_1 = A*exp(-alpha*sqrt( (s(1) - rx(1,2))^2 + (s(2) ...
        - rx(2,2))^2 + (s(3) - rx(3,2))^2 ));
```

```
    amp_2 = A*exp(-alpha*sqrt( (s(1) - rx(1,3))^2 + (s(2) ...
        - rx(2,3))^2 + (s(3) - rx(3,3))^2 ));
    varargout{1} = A*exp(-alpha*sqrt( (s(1) - rx(1,4))^2 + ...
        (s(2) - rx(2,4))^2 + (s(3) - rx(3,4))^2 ));
 end

end
```

get_AoA.m

```
function theta_est = get_AoA(d_pr, d_qr, t_pr, t_qr, v)
%%
% Function to estimate theta - Right Triangle case
% Input parameters ...
    ------------------------------------------------------------
%
% d_pr          : distance between P and R
% d_qr          : distance between Q and R
% t_pr          : time difference between P and R
% t_qr          : time difference between P and Q
% v             : sound speed
%
% Output parameters ...
    -----------------------------------------------------------
%
% theta_est     : theta estimate
%
%-----------------------------------------------------------------
% Calculating theta from t_pr
    theta_est_pr = real(acosd((abs(t_pr)*v)/abs(d_pr)));

% Calculating theta from t_qr
    theta_est_qr = real(asind((abs(t_qr)*v)/abs(d_qr)));

% Determining quadrant
    if(t_pr > 0 && t_qr > 0)
        % Quadrant I
        theta_est_qr = 90 - theta_est_qr;
        theta_est_pr = 90 - theta_est_pr;
    elseif(t_pr > 0 && t_qr < 0)
        % Quadrant II
        theta_est_qr = 90 + theta_est_qr;
        theta_est_pr = 90 + theta_est_pr;
    elseif(t_pr < 0 && t_qr < 0)
        % Quadrant III
        theta_est_qr = 270 - theta_est_qr;
```

```
        theta_est_pr = 270 - theta_est_pr;
    elseif(t_pr < 0 && t_qr > 0)
        % Quadrant IV
        theta_est_qr = 270 + theta_est_qr;
        theta_est_pr = 270 + theta_est_pr;
    end

 theta_est = mean([theta_est_pr theta_est_qr]);

end
```

get_SoundSpeed.m

```
function [sound_speed, alpha] = get_SoundSpeed(T, P, RH, f)
%%
% Function to obtain speed of sound
% Input parameters ...---------------------------------------------------------
%
% T            : room temperature in degree C
% P            : pressure in Pascals
% RH           : relative humidity as fraction. 0 ≤ RH ≤ 1.
%
% Output parameters ...--------------------------------------------------------
%
% alpha        : attenuation parameter
% velo         : velocity of sound in air
%
%------------------------------------------------------------------------------

 % Calculating viscosity (Sutherland's Law of Viscosity)
 eta = (1.716e-5 * (273.15/T+273.15)^(3/2)) * (383.55/(T + ...
     383.55));

 % Calculating air density
    % Calculating water vapour pressure (Herman Wobus' ...
        Equation)
    c = [0.99999683 -0.90826951e-2 0.78736169e-4 ...
        -0.61117958e-6 0.43884187e-8 ...
        -0.29883885e-10 0.21874425e-12 -0.17892321e-14 ...
            0.11112018e-16 ...
        -0.30994571e-19];
    p = (c(1)+T*(c(2)+T*(c(3)+T*(c(4)+T*(c(5)+T*(c(6)...
    +T*(c(7)+T*(c(8)+T*(c(9)+T*(c(10))))))))));

```

```
    Pv = RH * (6.1078/(p^8));    % mb
    Pv = Pv*100;                 % pascal

Pd = P - Pv;
rho = (Pd/(287.05*(T+273.25))) + (Pv/(461.495*(T+273.25)));

% Calculating speed of sound
sound_speed = 331.1*sqrt(1 + (T/273.15));

% Calculating attenuation coefficient
alpha = (2*eta*(2*pi*f)^2)/(3*rho*(sound_speed^3));

end
```

get_TDoAEstimate.m

```
function [varargout] = get_TDoAEstimate(x_ref, F, varargin)
%%
% Function to obtain TDoA estimates using Hilbert Transform
% Input parameters ...
   -----------------------------------------------------------
%
% x_ref          : Signal wrt which time difference is to be ...
   calculated
% F              : Center frequency
% x_n            : nth signal
%
% *Note: * All the input signals are row vectors
%
% Output parameters ...
   ----------------------------------------------------------
%
% t_n            : time difference of arrival between ...
   reference signal and
%                  nth input signal
%-------------------------------------------------------------

 if(nargin < 1)
     disp('INVALID number of input signals');
     return;
 end

 if(nargout ≠ nargin-2)
    disp('Invalid number of input signals and output ...
       delays.');
    return;
```

```
    end

    varargout = cell(1,nargout);

    x_ref_ht = imag(hilbert(x_ref));
    b = (x_ref_ht.')/(x_ref_ht*(x_ref_ht.'));

    for k = 1:nargin-2
        % equation (3.16)
        varargout{k} = ...
                (1/(2*pi*F))*asin(-(varargin{k}*b));
    end
end
```

get_xyz.m

```
function[x,y,z]=get_xyz(tdoa_a,tdoa_b,tdoa_d,v,d)
%%
% Function to estimate source coordinates for square ...
    configuration of the
% sensors
% Input parameters ...
    ----------------------------------------------------------
%
% tdoa_a        : time difference between A and C
% tdoa_b        : time difference between B and C
% tdoa_d        : time difference between D and C
% d             : Distance between 2 adjacent sensors
% v             : sound speed
%
% Output parameters ...
    ---------------------------------------------------------
%
% x     : Estimated x-coordinate of the source
% y     : Estimated y-coordinate of the source
% z     : Estimated z-coordinate of the source
%
%-------------------------------------------------------------

%Calculating difference between the distance traveled by ...
    sound wave from
%source to sensor (From Eq. 4.4, 4.5, 4.6, 4.7)
D_ba=v*(tdoa_b-tdoa_a);
D_bd=v*(tdoa_b-tdoa_d);
D_ac=v*(tdoa_a);
D_dc=v*(tdoa_d);
```

```
%Calculating estimated coordinates of source
%(From Eq. 4.14, 4.15, 4.16)
x=D_ba*(D_dc-D_ac)*(d^2+D_ac*D_dc) - ...
    D_dc*(D_ba-D_bd)*(d^2-D_ba*D_bd);
x=x/(2*d*(D_bd*D_dc-D_ac*D_ba));
y=(D_dc-D_ac)*(d^2+D_dc*D_ac) + 2*d*D_ac*x;
y=y/(2*d*D_dc);
z=(d^2-2*d*x-D_dc^2)/(2*D_dc);
z=sqrt(z^2-x^2-y^2);
```

single_freq_filter.m

```
function x_filtered = single_freq_filter(x)
    N = length(x);                 % length of input sequence

    x_fft = fft(x);                 % performing fft

    max_val = max(abs(x_fft)); % finding the maximum ...
        amplitude present in fft

    % Replacing other frequencies with zero
    for k = 1:N
        if(abs(x_fft(k)) < max_val)
            x_fft(k) = 0;
        end
    end

    % converting back to time domain
    x_filtered = real(ifft(x_fft));

end
```

Chapter 8

Simulation Test

This chapter shows two example runs on the simulation toolkit developed as a part of this project

8.1 Simulation Run 1

```
Command Window
Welcome to ASL-SimTool

---ENVIRONMENTAL CONFIGURATION---

Describe the terrestrial environment
Room temperature (in degree C): 25
Absolute pressure (in Pascals | at sea-level:101.325e5) : 101.325e5
Relative humidity in between [0,1]: 0.6
Noise level (in dB): 20

---SENSOR CONFIGURATION---
Select your sensor configuration:
[1] Right triangle
[2] Square
Your option: 1
Sensor R is located at the origin (0,0)
Enter the x-coordinate of Q (in m): 0.12
Enter the y-coordinate of P (in m): 0.12
Enter the sampling frequency (in Hz): 8e3
Enter the buffer size (preferable power of 2): 1024

---SOURCE CONFIGURATION---
Enter the source frequency: 20
Enter the source amplitude: 10
Source function:
[1] Sine
[2] Cosine
Your option: 2
Enter the x-coordinate of S (in m): 23
Enter the y-coordinate of S (in m): -45

---SIMULATING---
Actual time delay:
t_pr = -0.000309
t_qr = 0.000158
Estimate time delay:
tau_est_p = -0.000244
tau_est_q = 0.000071
Actual angle: -62.927920
Estimated angle: 298.557841>> 360-298.557841

ans =

fx    61.4422
```

Figure 8.1: Simulating right triangle configuration

8.2 Simulation Run 2

```
Command Window
Welcome to ASL-SimTool

---ENVIRONMENTAL CONFIGURATION---

Describe the terrestrial environment
Room temperature (in degree C): 25
Absolute pressure (in Pascals | at sea-level:101.325e5) : 101.325e5
Relative humidity in between [0,1]: 0.5
Noise level (in dB): 3000

---SENSOR CONFIGURATION---
Select your sensor configuration:
[1] Right triangle
[2] Square
Your option: 2
Sensor C is located at the origin (0,0)
Since it is a square configuration distance between adjacent sensors is same
*A(0,d,0)    *B(d,d,0)

*C(0,0,0)    *D(d,0,0)
Enter distance between any two sensors, d(in m): 0.1
Enter the sampling frequency (in Hz): 10e3
Enter the buffer size (preferable power of 2): 1024

---SOURCE CONFIGURATION---
Enter the source frequency: 10
Enter the source amplitude: 8
Source function:
[1] Sine
[2] Cosine
Your option: 2
Enter the x-coordinate of S (in m): 3
Enter the y-coordinate of S (in m): 4
Enter the z-coordinate of S (in m): 5

---SIMULATING---
Actual time delay:
t_ac = -0.000162
t_bc = -0.000284
t_dc = -0.000121
Estimate time delay:
tau_est_ac = -0.000158
tau_est_bc = -0.000277
fx tau_est_dc = -0.000118
```

```
Estimate time delay:
tau_est_ac = -0.000158
tau_est_bc = -0.000277
tau_est_dc = -0.000118
x: 2.866151
y: 3.820779
fx z: 4.994270>>
```

Figure 8.2: Simulating square configuration

8.3 Observations and Inferences

- As mentioned previously in this report, the accuracy of the techniques depend on how good the filtering of a noisy signal has been done.
- The run cases show that with high SNR, the simulation provides good estimate.

Chapter 9

Conclusion

In this project, we have discussed multiple methods of TDoA estimation, source localization, and also provided two methods to perform the same. However, these methods are ad-hoc in nature and their performance is also discussed. The localization algorithms heavily depend on how good the noise is filtered. The filter must be designed to eliminate phase distortion introduced due noise. We have presented a modular simulation toolkit - **ASL-SimTool** which can be used during the design phase of the hardware for acoustic source localization.

9.1 Future Work

We plan to implement the methods discussed in the report on hardware.

- For better estimation, the signals must be filtered to a SNR level of 20 dB. Work needs to be done on the filtering part. The filter can be of analog and/or digital nature.
- Better TDoA estimation methods which provide good results in noisy environments can be explored.

Appendix A

This appendix contains the MATLAB implementation for

- Normalized frequency based TDoA implementation *[Go to code]*
- GCC-PHAT based TDoA implementation *[Go to code]*
- Hilbert Transform based TDoA implementation *[Go to code]*
- Right triangle configuration - Angle of arrival *[Go to code]*

Normalized frequency based TDoA implementation

```
% Time difference estimation based on Normalized frequency

close all
clear
clc

%% Signal and sensor parameters

F = 10;                   % source frequency (Hz)
A = 5;                    % source amplitude
alpha_1 = 0.7;            % attenuation at reference sensor
alpha_2 = 0.71;           % attenuation at other sensor
src_angs = 0:0.5:360;     % angle made by source w.r.t. ...
    x-axis (degrees)
v = 340;                  % speed of sound in air (m/s)

% Both the sensors are assumed to lie on x-axis
d = 0.15;                 % distance between the sensors ...
    (metres)
sample_time = 0.5;        % time for which signal should be ...
    sampled (secs)
```

```
Fs = 10e3;                % sampling frequency (Hz)
N = 1024;                 % buffer size (total number of ...
    samples to consider)

SNR = -40:10:10;            % signal to noise ratio (dB)

%% Generation of signal

n = 0:1/Fs:sample_time;            % generating sample ...
    instances
if (length(n)< N)
    disp('Samples are less than buffer. Modify the buffer, ...
        Fs, or sample time.');
    return;
end
n = n(1:N);

t_diff = d*cosd(src_angs)/v;        % from equation (3.2)

x_1 = alpha_1*A*cos(2*pi*F*n);      % generating reference ...
    signal

%% Estimating time difference

tau_no_noise_est = zeros(size(t_diff));                       ...
    % array to store estimates without noise
tau_noise_est = zeros(length(SNR),length(t_diff));            ...
    % array to store estimates with noise

f_0 = F/Fs;                                   % from equation (3.5)

t_diff_idx = 1;
% Without noise
for tau=t_diff
   x_2 = alpha_2*A*cos(2*pi*F*(n + tau));

   X_1 = 0j;
   X_2 = 0j;

   % equation (3.6)
   for m = 1:N
       X_1 = X_1 + x_1(m)*exp(-1j*2*pi*f_0*(m-1));
       X_2 = X_2 + x_2(m)*exp(-1j*2*pi*f_0*(m-1));
   end

   % equation (3.7)
   tau_no_noise_est(t_diff_idx) = (angle(X_2) - ...
       angle(X_1))/(2*pi*F);
   t_diff_idx = t_diff_idx + 1;
```

```
end

% Observing time difference estimations by varying noise ...
    levels

n_1 = randn(1,N);                % noise at sensor 1
n_2 = randn(1,N);                % noise at sensor 2

t_diff_idx = 1;
for tau=t_diff
   x_2 = alpha_2*A*cos(2*pi*F*(n + tau));

   snr_idx = 1;
   for snr = SNR
     x_1_noise = x_1 + ((norm(x_1)/norm(n_1)) * ...
         10^(-snr/20))*n_1;
     x_2_noise = x_2 + ((norm(x_2)/norm(n_2)) * ...
         10^(-snr/20))*n_2;

     X_1 = 0j;
     X_2 = 0j;

     for m = 1:N
         X_1 = X_1 + x_1_noise(m)*exp(-1j*2*pi*f_0*(m-1));
         X_2 = X_2 + x_2_noise(m)*exp(-1j*2*pi*f_0*(m-1));
     end

     tau_noise_est(snr_idx, t_diff_idx) = (angle(X_2) - ...
         angle(X_1))/(2*pi*F);
     snr_idx = snr_idx + 1;
   end
   t_diff_idx = t_diff_idx + 1;
end

%% Plotting results

% Plotting actual and estimated time difference (no noise)
figure;
plot(src_angs,t_diff,src_angs,tau_no_noise_est);
xlim([0 360]);
title('Estimate and actual time difference (no noise)');
legend('actual delay','estimated delay');
xlabel('Source angle (degrees)');
ylabel('Time difference (seconds)');
grid on;

% Plotting actual and estimated time difference (with noise)
figure;
```

```
sgtitle('Estimate and actual time difference (with varying ...
    noise levels)');
for k = 1:length(SNR)
    subplot(3,2,k);
    plot(src_angs,t_diff,src_angs,tau_noise_est(k,:));
    xlim([0 360]);
    legend('actual delay',strcat(num2str(SNR(k)),' dB'));
    title(strcat('SNR: ',num2str(SNR(k)),' dB'));
    xlabel('Source angle (degrees)');
    ylabel('Time difference (seconds)');
    grid on;
end
```

GCC-PHAT based TDoA implementation

```
% Time difference estimation based on GCCPHAT

close all
clear
clc

%% Signal and sensor parameters

F = 10;                  % source frequency (Hz)
A = 5;                   % source amplitude
src_angs = 0:180;        % angle made by source w.r.t. ...
    x-axis (degrees)
v = 340;                 % speed of sound in air (m/s)
SNR = -40:10:10;         % signal to noise ratio (dB)

% Both the sensors are assumed to lie on x-axis
d = 0.15;                % distance between the sensors ...
    (metres)
sample_time = 0.5;       % time for which signal should be ...
    sampled (secs)
Fs = 10e3;               % sampling frequency (Hz)
N = 1024;                % buffer size (total number of ...
    samples to consider)

mic = phased.OmnidirectionalMicrophoneElement;
array = phased.ULA(2,d,'Element',mic);

%% Generation of signal and estimation

```

```
n = 0:1/Fs:sample_time;            % generating sample ...
    instances
if (length(n)< N)
    disp('Samples are less than buffer. Modify the buffer, ...
        Fs, or sample time.');
    return;
end
n = n(1:N);

Ncorr = 2*length(n) - 1;      % length of cross-correlation ...
    sequence
Nfft = 2^nextpow2(Ncorr);         % length of FFT
p = -(Ncorr-1)/2:(Ncorr-1)/2;     % lags of cross-correlation
p = p/Fs;

% Sensors lie along y-axis
t_diff = d*sind(src_angs)/v;      % from equation (3.2)
tau_noise_est = zeros(length(SNR),length(src_angs)); % ...
    array to store estimates without noise

% Without noise
t_diff_idx = 1;
for aoa = src_angs
    snr_idx = 1;
    for snr = SNR
        s_sig = awgn(A*cos(2*pi*F*n),snr,'measured');
        collector = phased.WidebandCollector('Sensor',array...
        ,'PropagationSpeed',v,...
        'SampleRate',Fs,'ModulatedInput',false);
        signal = collector(s_sig.',[aoa;0]);

        % equation (3.12)
        X_1 = fft(signal(:,1).',Nfft);
        X_2 = fft(signal(:,2).',Nfft);
        csp = X_2.*conj(X_1);                          % Cross ...
            power spectrum
        % equation (3.11)
        r_hat_x1x2_temp = fftshift(ifft(exp(-1j*angle(csp))));
        r_hat_x1x2 = r_hat_x1x2_temp(Nfft/2+1...
        -(Ncorr-1)/2:Nfft/2+1+(Ncorr-1)/2);

        [¬, idx] = max(r_hat_x1x2);
        % equation (3.10)
        tau_noise_est(snr_idx,t_diff_idx) = p(idx);

        snr_idx = snr_idx + 1;
    end
     t_diff_idx = t_diff_idx + 1;
end
```

```

%% Plotting results

% Plotting actual and estimated time difference (with noise)
figure;
sgtitle('Estimate and actual time difference (with varying ...
    noise levels)');
for k = 1:length(SNR)
    subplot(3,2,k);
    plot(src_angs,t_diff,src_angs,tau_noise_est(k,:));
    xlim([0 360]);
    legend('actual delay',strcat(num2str(SNR(k)),' dB'));
    title(strcat('SNR: ',num2str(SNR(k)),' dB'));
    xlabel('Source angle (degrees)');
    ylabel('Time difference (seconds)');
    grid on;
end
```

Hilbert Transform based TDoA implementation

```
% Time difference estimation based on Hilbert transform

close all
clear
clc

%% Signal and sensor parameters

F = 10;                 % source frequency (Hz)
A = 5;                  % source amplitude
alpha_1 = 0.7;          % attenuation at reference sensor
alpha_2 = 0.71;         % attenuation at other sensor
src_angs = 0:0.5:360;   % angle made by source w.r.t. ...
    x-axis (degrees)
v = 340;                % speed of sound in air (m/s)

% Both the sensors are assumed to lie on x-axis
d = 0.15;               % distance between the sensors ...
    (metres)
sample_time = 0.5;      % time for which signal should be ...
    sampled (secs)
Fs = 10e3;              % sampling frequency (Hz)
```

```matlab
N = 1024;                  % buffer size (total number of ...
    samples to consider)

SNR = -40:10:10;           % signal to noise ratio (dB)

%% Generation of signal

n = 0:1/Fs:sample_time;            % generating sample ...
    instances
if (length(n)< N)
    disp('Samples are less than buffer. Modify the buffer, ...
        Fs, or sample time.');
    return;
end
n = n(1:N);

t_diff = d*cosd(src_angs)/v;       % from equation (3.2)

x_1 = alpha_1*A*cos(2*pi*F*n).';      % generating ...
    reference signal

% Taking Hilbert transform of reference signal --> x_1
% hilbert(sig_r)= sig_r + j(HT{sig_r}) ---> Analytic ...
    signal of sig_r, where
% HT{} is hilbert transform
% Hence taking imag(hilbert(sig_r)) returns hilbert ...
    transform of the signal
x_1_ht=imag(hilbert(x_1));

%% Estimating time difference

tau_no_noise_est = zeros(size(t_diff));                    ...
    % array to store estimates without noise
tau_noise_est = zeros(length(SNR),length(t_diff));         ...
    % array to store estimates with noise

t_diff_idx = 1;
% Without noise
for tau=t_diff
   x_2 = alpha_2*A*cos(2*pi*F*(n + tau)).';

   % equation (3.16)
   tau_no_noise_est(t_diff_idx) = ...
     (1/(2*pi*F))*asin(-((x_2.')*x_1_ht)/((x_1.')*x_1));

   t_diff_idx = t_diff_idx + 1;
end

```

```
% Observing time difference estimations by varying noise ... levels

n_1 = randn(1,N).';                 % noise at sensor 1
n_2 = randn(1,N).';                 % noise at sensor 2

t_diff_idx = 1;
for tau=t_diff
   x_2 = alpha_2*A*cos(2*pi*F*(n + tau)).';

   snr_idx = 1;
   for snr = SNR
     x_1_noise = x_1 + ((norm(x_1)/norm(n_1)) * ... 10^(-snr/20))*n_1;
     x_2_noise = x_2 + ((norm(x_2)/norm(n_2)) * ... 10^(-snr/20))*n_2;

     x_1_ht_noise=imag(hilbert(x_1_noise));

     % equation (3.16)
     tau_noise_est(snr_idx, t_diff_idx) = ...
 (1/(2*pi*F))*asin(-((x_2_noise.')*x_1_ht_noise)/...
 ((x_1_noise.')*x_1_noise));

     snr_idx = snr_idx + 1;
   end
   t_diff_idx = t_diff_idx + 1;
end

%% Plotting results

% Plotting actual and estimated time difference (no noise)
figure;
plot(src_angs,t_diff,src_angs,tau_no_noise_est);
xlim([0 360]);
title('Estimate and actual time difference (no noise)');
legend('actual delay','estimated delay');
xlabel('Source angle (degrees)');
ylabel('Time difference (seconds)');
grid on;

% Plotting actual and estimated time difference (with noise)
figure;
sgtitle('Estimate and actual time difference (with varying ... noise levels)');
for k = 1:length(SNR)
    subplot(3,2,k);
    plot(src_angs,t_diff,src_angs,tau_noise_est(k,:));
    xlim([0 360]);
```

```
    legend('actual delay',strcat(num2str(SNR(k)),' dB'));
    title(strcat('SNR: ',num2str(SNR(k)),' dB'));
    xlabel('Source angle (degrees)');
    ylabel('Time difference (seconds)');
    grid on;
end
```

Right triangle configuration - Angle of arrival

```
function theta_est = getAngleOfArrival(d_pr, d_qr, t_pr, ...
    t_qr, v)
% Calculating theta from t_pr
    theta_est_pr = real(acosd((abs(t_pr)*v)/abs(d_pr)));

% Calculating theta from t_qr
    theta_est_qr = real(asind((abs(t_qr)*v)/abs(d_qr)));

% Determining quadrant
    if(t_pr > 0 && t_qr > 0)
        % Quadrant I
        theta_est_qr = 90 - theta_est_qr;
        theta_est_pr = 90 - theta_est_pr;
    elseif(t_pr > 0 && t_qr < 0)
        % Quadrant II
        theta_est_qr = 90 + theta_est_qr;
        theta_est_pr = 90 + theta_est_pr;
    elseif(t_pr < 0 && t_qr < 0)
        % Quadrant III
        theta_est_qr = 270 - theta_est_qr;
        theta_est_pr = 270 - theta_est_pr;
    elseif(t_pr < 0 && t_qr > 0)
        % Quadrant IV
        theta_est_qr = 270 + theta_est_qr;
        theta_est_pr = 270 + theta_est_pr;
    end

 theta_est = mean([theta_est_pr theta_est_qr]);

end
```

```
% Estimating angle of arrival

close all
```

```
clear
clc

%% Signal and sensor parameters

F = 10;                    % source frequency (Hz)
A = 5;                     % source amplitude
alpha_r = 0.7;             % attenuation at reference sensor
alpha_p = 0.71;            % attenuation at P sensor
alpha_q = 0.705;           % attenuation at Q sensor
src_angs = 0:0.5:360;      % angle made by source w.r.t. ...
    x-axis (degrees)
v = 340;                   % speed of sound in air (m/s)

% Both the sensors are assumed to lie on x-axis
d = 0.15;                  % distance between the sensors ...
    (metres)
sample_time = 0.5;         % time for which signal should be ...
    sampled (secs)
Fs = 10e3;                 % sampling frequency (Hz)
N = 1024;                  % buffer size (total number of ...
    samples to consider)
SNR = [-10,15,20,30];                  % signal to noise ...
    ratio (dB)

%% Generation of signal

n = 0:1/Fs:sample_time;            % generating sample ...
    instances
if (length(n)< N)
    disp('Samples are less than buffer. Modify the buffer, ...
        Fs, or sample time.');
    return;
end
n = n(1:N);

t_diff = [d*sind(src_angs)/v;d*cosd(src_angs)/v];         % ...
    from equation (3.1),(3.2)

x_r = alpha_r*A*cos(2*pi*F*n).';      % generating ...
    reference signal

x_r_ht=imag(hilbert(x_r));

%% Estimating angle of arrival

% array to store tdoa estimates
tau_est = zeros(size(t_diff));

```

```
% array to store aoa estimates with and without noise
theta_est = zeros(1,length(src_angs));

% Without noise
for t_diff_idx = 1:length(src_angs)
   x_p = alpha_p*A*cos(2*pi*F*(n + t_diff(1,t_diff_idx))).';
   x_q = alpha_q*A*cos(2*pi*F*(n + t_diff(2,t_diff_idx))).';

   % equation (3.16)
   tau_est(1,t_diff_idx) = ...
(1/(2*pi*F))*asin(-((x_p.')*x_r_ht)/((x_r.')*x_r));

   tau_est(2,t_diff_idx) = ...
(1/(2*pi*F))*asin(-((x_q.')*x_r_ht)/((x_r.')*x_r));

   % equation 4.1
   theta_est(t_diff_idx) = ...
    getAngleOfArrival(d, d, tau_est(1,t_diff_idx),...
        tau_est(2,t_diff_idx), v);
end

% Plotting actual and estimated time difference (no noise)
figure;
plot(src_angs,src_angs,src_angs,theta_est);
xlim([-2.5 360]);
title('Estimate and actual AoA (no noise)');
legend('actual AoA','estimated AoA');
xlabel('angle (degrees)');
ylabel('angle (degrees)');
grid on;

% With noise
n_r = randn(1,N).';                % noise at sensor R
n_p = randn(1,N).';                % noise at sensor P
n_q = randn(1,N).';                % noise at sensor Q

% array to store filtered signal
x_filtered = zeros(3,length(n));

snr_idx = 1;                     % var for subplots
figure;
for snr = SNR
   for t_diff_idx = 1:length(src_angs)
     x_p = alpha_p*A*cos(2*pi*F*(n + t_diff(1,t_diff_idx))).';
     x_q = alpha_q*A*cos(2*pi*F*(n + t_diff(2,t_diff_idx))).';

     x_r_noise = x_r + ((norm(x_r)/norm(n_r)) * ...
        10^(-snr/20))*n_r;
     x_p_noise = x_p + ((norm(x_p)/norm(n_p)) * ...
```

```
            10^(-snr/20))*n_p;
        x_q_noise = x_q + ((norm(x_q)/norm(n_q)) * ...
            10^(-snr/20))*n_q;

        x_filtered(1,:) = single_freq_filter(x_r_noise.');
        x_filtered(2,:) = single_freq_filter(x_p_noise.');
        x_filtered(3,:) = single_freq_filter(x_q_noise.');

        x_r_ht = imag(hilbert(x_filtered(1,:)));

        b = (x_r_ht.')/(x_filtered(1,:)*(x_filtered(1,:).'));

        % equation (3.16)
        tau_est(1,t_diff_idx) = ...
                (1/(2*pi*F))*asin(-(x_filtered(2,:)*b));

        tau_est(2,t_diff_idx) = ...
                (1/(2*pi*F))*asin(-(x_filtered(3,:)*b));

        % equation 4.1
        theta_est(t_diff_idx) = ...
        getAngleOfArrival(d, d, tau_est(1,t_diff_idx),...
                            tau_est(2,t_diff_idx), v);
    end

    % Plotting actual and estimated angle of arrival (with ...
        noise)
    subplot(1,length(SNR),snr_idx);
    plot(src_angs,src_angs,src_angs, theta_est);
    xlim([-2.5 360]);
    title(strcat('Estimate and actual AoA  ',...
            strcat('(noise: ',num2str(snr),' dB)')));
    legend('actual AoA','estimated AoA');
    xlabel('angle (degrees)');
    ylabel('angle (degrees)');
    grid on;

    snr_idx = snr_idx + 1;
end
```

Appendix B

This appendix discusses about the filter we devised and used in this project. The filter described here gives the same output as given by a standard lowpass filter. This filter can be replaced with a lowpass or a bandpass filter if the source frequency has a wider bandwidth. The chapter also has listing of the function implementing the filtering algorithm discussed and a script to test it's output.

Filtering algorithm

The filtering performed is not in a conventional sense. It is based on the fact that DFT magnitude plot of a noisy sequence gives a peak at frequencies present within the signal. Since in this project we use only a single frequency signal, this algorithm works well. The steps performed are:

1. Take DFT of the sequence
2. Identify the peaks
3. Replace all frequencies except the one's at which peak exist with zero
4. Perform IDFT to obtain the filtered signal.

Implementation Results

Same noisy signal was passed through the custom filter and a Butterworth lowpass filter.

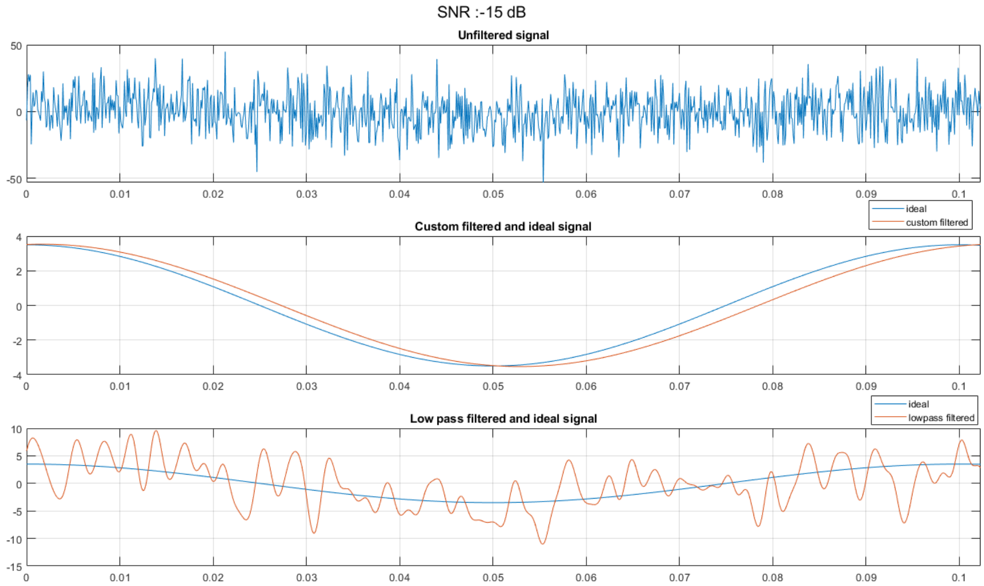


Figure 9.1: Output of custom filter

Observations and Inferences

- The output from the custom filter is similar to the ideal waveform but it is phase shifted due to effect of noise on phase of the signal.
- The lowpass filter output has frequency components other than the center frequency.
- After performing multiple tests on the filters by varying SNR levels, we found that the custom filter works good only for SNR > -15 dB. For low SNR levels, we cannot use this.

The design of a filter block prior to TDoA estimation requires a separate research and base altogether. In this project, we do not aim to present the best filtering method, but use a convinient filter to achieve our goal of localization.

MATLAB Code

The filter function has been listed here - *[Goto code]*

Test script

```
% Testing custom filter

close all
clear
clc

%% Signal and sensor parameters

F = 10;                  % source frequency (Hz)
A = 5;                   % source amplitude
alpha_1 = 0.7;           % attenuation at reference sensor

% Both the sensors are assumed to lie on x-axis
d = 0.15;                % distance between the sensors ...
    (metres)
sample_time = 0.5;       % time for which signal should be ...
    sampled (secs)
Fs = 10e3;               % sampling frequency (Hz)
N = 1024;                % buffer size (total number of ...
    samples to consider)
SNR = -15;

%% Generation of signal

n = 0:1/Fs:sample_time;            % generating sample ...
    instances
if (length(n)< N)
    disp('Samples are less than buffer. Modify the buffer, ...
        Fs, or sample time.');
    return;
end
n = n(1:N);

x_1 = alpha_1*A*cos(2*pi*F*n).';      % generating ...
    reference signal

%% Filtering

x_1_noise = awgn(x_1, SNR, 'measured');
x_filtered = single_freq_filter(x_1_noise.');

x_filtered_low=lowpass(x_1_noise,F,Fs);

%% Plotting
sgtitle(strcat('SNR :',num2str(SNR),' dB'));
```

```

subplot(3,1,1);
plot(n,x_1_noise);
title('Unfiltered signal');
xlim([min(n) max(n)]);
grid on;

subplot(3,1,2);
plot(n,x_1,n,x_filtered);
title('Custom filtered and ideal signal');
legend('ideal','custom filtered');
xlim([min(n) max(n)]);
grid on;

subplot(3,1,3);
plot(n,x_1,n,x_filtered_low);
title('Low pass filtered and ideal signal');
legend('ideal','lowpass filtered');
xlim([min(n) max(n)]);
grid on;
```

Bibliography

[1] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall Press, USA, 3rd edition, 2009.

[2] Dan E. Dudgeon Don H. Johnson. *Array Signal Processing Concepts and Techniques*. Prentice Hall Signal Processing Series, 1993.

[3] Murry L Salby. *Fundamentals of Atmospheric Physics*. Elsevier, 1996.

[4] K Scarbrough, N Ahmed, and G Clifford Carter. Comparison of two methods for time delay estimation of sinusoids. *Proceedings of the IEEE*, 70(1):90–92, 1982.

[5] G. Clifford Carter Charles H. Knapp. The generalized correlation method for estimation of time delay. *IEEE Transactions on Acoustics, Speech and Singal Processing*, 24(4):320–327, 1976.

[6] G Clifford Carter. Coherence and time delay estimation. *Proceedings of the IEEE*, 75(2):236–255, 1987.

[7] Hilman Adritya HBB Catur and Hendri Maja Saputra. Azimuth estimation based on generalized cross correlation phase transform (gcc-phat) using equilateral triangle microphone array. In *2019 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)*, pages 89–93. IEEE, 2019.

[8] Bert Van Den Broeck, Alexander Bertrand, and Peter Karsmakers. Time-domain gcc-phat sound source localization for small microphone arrays.

[9] J Hassab and R Boucher. Optimum estimation of time delay by a generalized correlator. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(4):373–380, 1979.

[10] Anders Grennberg and Magnus Sandell. Estimation of subsample time delay differences in narrowband ultrasonic echoes using the hilbert transform correlation. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 41(5):588–595, 1994.

[11] Hoang Do, Harvey F Silverman, and Ying Yu. A real-time srp-phat source location implementation using stochastic region contraction (src) on a large-aperture microphone array. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 1, pages I–121. IEEE, 2007.

[12] Dimitris G. Manolakis John G. Proakis. *Digital Signal Processing: Principles, Algorithms, and Applications 4th edition*, pages 21–22. Pearson Prentice Hall, 2006.

[13] MANUEL T. SILVIA. Chapter 11 - time delay estimation. In Douglas F. Elliott, editor, *Handbook of Digital Signal Processing*, pages 789 – 855. Academic Press, San Diego, 1987.

[14] Yiteng Huang Jacob Benestsy, Jingdong Chen. *Microphone Array Signal Processing.* Springer, 2008.

[15] Bo Qin, Heng Zhang, Qiang Fu, and Yonghong Yan. Subsample time delay estimation via improved gcc phat algorithm. In *2008 9th International Conference on Signal Processing*, pages 2579–2582. IEEE, 2008.

[16] R.N. Bracewell. *The Fourier transform and its applications, 2 ed.* New York:McGraw-Hill, 1986.