Fast Hardware-Oriented Algorithms for 3D Positioning in LOS and Single Bounced NLOS Environments

Cem Yağlı

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Applied Mathematics and Computer Science

Eastern Mediterranean University
June 2021
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Ali Hakan Ulusoy
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy in Applied Mathematics and Computer Science.

Prof. Dr. Nazım Mahmudov
Chair, Department of Mathematics

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Doctor of Philosophy in Applied Mathematics and Computer Science.

Asst. Prof. Dr. Emre Özen
Co-Supervisor

Assoc. Prof. Dr. Arif Akkeleş
Supervisor

Examining Committee

1. Prof. Dr. Rashad Aliyev
2. Prof. Dr. Serhan Dağtaş
3. Prof. Dr. Hasan Demirel
4. Prof. Dr. Hayri Sever
5. Assoc. Prof. Dr. Arif Akkeleş
6. Asst. Prof. Dr. Emre Özen
7. Asst. Prof. Dr. Mehmet Ali Tut

ABSTRACT

The ability to find the location of a mobile object and track it in two-dimensional (2D) and three-dimensional (3D) space is an indispensable feature of wireless communication systems. These studies have been supported by governments and businesses since the beginning. In particular, the increase in demand for using self-navigation technology in unmanned vehicles is attracting additional attention to this area of research. The self-location estimation process is an indefeasible capability that an unmanned vehicle should have. This process requires high accuracy that is hard to be achieved by mobile devices, usually having limited power and computing capabilities. The methods and techniques developed for these systems compete in terms of simplicity, performance, and accuracy. However, the majority of solutions focus on only one of these performance metrics and are not applicable to the projected systems.

In this study, two new location estimation solutions that satisfy all three performance metrics (simplicity, accuracy, and efficiency) are developed. Accordingly, two accelerated variations of Coordinate Rotation Digital Computer (CORDIC) algorithms are used for the first time with line-of-sight (LOS) and single-bounced-scattering non-line-of-sight (SBS NLOS) signal arriving approaches in the self-positioning process. The simplicity of the mathematical operations used, namely binary shift and add operations, makes our solutions hardware friendly.

The second novelty of the study is the newly developed “Vector Breaking Method” (VBM) used for estimating the location of mobile objects using the arriving signals Bbounced from scatters with unknown locations.

Test results using the developed algorithms show that the projected level of accuracy has been achieved. Furthermore, our solutions have presented better performance than the solutions currently available in the field.

Keywords: 3D positioning, location estimation, LOS, line of sight, NLOS, non-line of sight, CORDIC, fast rotation algorithm, AOA, AOD, TOA, TDOA, EOTDO, CID, eCID, RSS, cellular wireless communication systems, One-bounced-scattering, hardware-oriented, antenna arrays, mobile object, base station, scatter, ACE algorithm, fixed angle rotation method, FARM, dynamic angle rotation method, DARM, vector breaking method, VBM, vector lengthening method, VLM, level of accuracy, computational cost, accuracy level.

ÖZ

Kablosuz iletişim sistemleri vasıtasıyla, iki boyutlu ve üç boyutlu uzayda hareket halindeki cihazların yerlerinin saptanabilmesi, takiplerinin yapılabilmesi son yıllarda en çok dikkat çeken teknolojilerdendirler. Bunların geliştirilmesine yönelik yapılan araştırmalar da devletler ve iş dünyası tarafından daha ilk günlerden beri desteklenmektedirler. Özellikle, insansız hava araçlarının kullanımının oldukça yaygınlaştığı günümüzde, hareket halindeyken bu araçlara kendi konumunu tespit edebilme yeteneğini kazandırmaya yönelik araştırmalar oldukça ilgi görmektedirler. Bugün, kendi konumunu bilerek seyir görevini yapabilmek, yeni nesil tüm insansız araçların vazgeçilemez bir yeteneği olmuştur. Sınırlı enerji ve işemci gücüne sahip bu tür hareketli araçlar, yüksek hassasiyet gerektiren bu tür işlemleri yerine getirmekte oldukça zorlanmaktadırlar. Bu amaçla geliştirilmiş bir çok çözüm yöntemi işlem basitliği, performans ve hassasiyet kıstaslarına göre birbirleriyle yarışmaktadırlar. Bu yöntemlerin büyük çoğunluğu aynı anda bu üç kıstası da sağlayamadıklarından, arzu edilen performansı gösteremiyorlar.

Bu tez çalışmamızda üç boyutlu uzayda hareket eden cihazların konum tespiti ve takibinde, aynı anda basitlik, hassasiyet ve verimlilik kıstaslarının üçünü de sağlayabilecek; sınırlı güç ve işlemci kapasitesiyle de kullanılabilecek çözümler üretmeyi hedefledik. Çözümümüz, yer tayini hesaplarının baz istasyonları üzerinde değil de hareketli cisim üzerinde yapılacağı şekilde tasarlandı. Bu amaçla, sadece direk olarak baz istasyonlarından gönderilen sinyallerle (LOS) değil, yeri belli olmayan engenlere vurup yansıyarak hareketli cihazımıza ulaşan (NLOS) sinyallerin de kullanılacağı bir grafiksel çözüm yöntemi geliştirildi. Ardından, hesaplarımızda

kullanılması gereken çok fazla işlemci gücü ve enerjisi gerektiren trigonometric fonksiyonları hızlandırılmış CORDIC algoritmaları ile değiştirerek, çözümlerimizi normalden daha hızlı ve az işlemci gücü ile de çalışabilmesi sağlandı.

Çalışmamız, CORDIC algoritmalarının ilk kez NLOS sinyalleri ile kullanılmasından dolayı eşsiz bir çalışma olarak değerlendirilebilir. Çalışmamızdaki ikinci özgünlük ise, ilk kez çalışmamızda geliştirilmiş ve kullanılmış olan "Vektör Kırma" Metodumuzdur (VBM). Bu metot yardımıyla, yerleri belli olmayan engellerden yansıyarak hareketli cismimize ulaşan baz istasyonu sinyallerini kullanılarak hareketli cismin yeri tahmin edilmektedir.

Çalışmamızda geliştirdiğimiz yer tayin yöntemleri simulasyonlarla test edildi ve hedeflenen doğrulukta sonuç verdikleri görüldü. Ayrıca, her iki yöntemimiz de, işlem ve enerji verimliliği konusunda, alanda kabul görmüş rakiplerinden daha iyi performanslar gösterdiler.

Anahtar Kelimeler: 3B konumlandırma, konum tahmini, direk görüş hattı, , indirek görüş hattı, CORDIC, hızlı açı döndürme algoritması, sinyal geliş açısı, sinyal gönderim açısı, sinyal geliş zamanı, sinyal geliş zaman farkı, yayın hücresi kimliği, hücresel kablosuz iletişim sistemleri, tek sekmeli yansıma, donanım uyumlu, anten dizisi, hareketli obje, baz 'stasyonu, engel, ACE algoritması, sabit açılı döndürme metodu, , değişken açılı döndürme metodu, vektör kırma metodu, vektör uzatma metodu, doğruluk seviyesi, işlemsel maliyet.

To My Family

ACKNOWLEDGEMENT

I would like to take this occasion to thank both, my supervisor Assoc. Prof. Dr. Arif Akkeleş and my co-supervisor Assist. Prof. Dr. Emre Özgen for their guidance, patience and motivation. In long meeting hours, Dr. Akkeleş helped me to understand, interpret and compare the alternative mathematical solutions while I was doing a literature review. He had also a great role on building the mathematical model of our solution. The thesis is steered by Dr. Özen, every step, every process was planned, controlled and guided by him intently. On building the computational models of our solution, designing algorithms, coding them in MATLAB and interpreting the simulation results; we spent months together.

On this long painful expedition, my family never made me feel alone. In every situation, they had encouraged me to keep going, they destressed me and always kept me on the way with their love, patient and supports. I would like to honour my family with this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ACE	Arif-Cem-Emre Algorithm
AOA	Angle of Arrival
AOD	Angle of Departure
BS	Base station
BTS	Solution of Behailu and Tobias
CID	Cell Identification
CORDIC	Coordinate Rotation Digital Computer
DARM	Dynamic Angle Rotation Method
eCID	Enhanced Cell Identification
FARM	Fixed Angle Rotation Method
FCC	Federal Communication Commission
GPS	Global Positioning System
MO	Mobile Object
OTDOA	Observed Time Difference of Arrival
RSS	Received Signal Strength
SAGE	Space Alternating Generalized Expectation
SBS NLOS	Single Bounced Scattering Non-Line of Sight
TDOA	Time Difference of Arrival
TOA	Time of Arrival
TOF	Time-of-Flight
TOT	Time-of-Transmission
UTDOA	Uplink Time Difference of Arrival

Chapter 1

INTRODUCTION

1.1 Introduction

Estimating the location of a mobile object (MO) is a remarkable research area for the last decades. Especially, having the ability of self-localisation has been a crucial necessity for MOs. Many applications stand on this feature are already introduced and has taken a place in our life. New researches on this area are grabbing the attention of governments and businesses as they are also the main financers of these studies. Federal Communication Commission (FCC) is the main authority that is dictating the E911 error margin accuracy requirements since 2001. Electronic bracelets are used to track down the infected people and those in contact with them now as we are living with Coronavirus disease 2019 (COVID-19). Through these applications, units of emergency vehicles, taxis, shuttles, and others are managed efficiently. Self-localisation is an indefeasibly part of the navigation systems of unmanned vehicles and smart missiles. This ability allows them to be operated on in any condition.

1.2 Location Estimation Through Wireless Network Signals

Although, the majority of existing systems are using Global Positioning Signals (GPS) in location estimation processes, in dense residential areas and tower blocks, the more powerful signals of cellular networks produce better results. Furthermore, considering the national security issues, GPS signals are not preferred to be used in various military operations. In many applications, wireless network signals are used in localisation processes broadcasted by fixed or portable base stations (BS).

1.3 Self-Positioning vs Remote Positioning Solutions

Especially, the self-navigation processes of the drones and smart missiles require precise and rapid self-localisation techniques which should simultaneously be energy efficient since these devices are mostly having limited power and processing capabilities. In remote positioning solutions, the location of MO is estimated on BS thru the signals transmitted by MO. Thereafter, this information is sent to MO with additional networking and time cost. Hence, self-localisation techniques are more efficient and applicable to the MO that is having limited resources.

1.4 Graphical Models and Calculation Techniques

There are many calculation techniques that are standing on various graphical solution models that are developed and competing for location estimation processes. Precision and efficiency (speed and cost of computations) are frequently used metrics in their comparisons. Unfortunately, very few solutions are satisfying the requirements in two metrics.

1.5 Objectives of the Thesis

The intend of this thesis is to develop a competitive self-positioning technique for MO as it must also achieve the following requirements:

- It must use wireless networking signals.
- It should work with both line-of-sight (LOS) and single-bounced-scattering non-line-of-sight (SBS NLOS) signal arriving approaches.
- All required calculations must be realised via the mobile object instead of the base stations.
- The developed model should be simple and computationally cost-effective enough as it is applicable to mobile devices even, they have very limited resources.

- It must be rapid and precise enough for satisfying the requirements of unmanned vehicles and smart missiles.

1.6 Contribution of the Thesis

To achieve the objectives, a unique graphical solution model is constructed that allows MO to self-position using both LOS and SBS NLOS approaches. The coordinate rotation digital computer (CORDIC) algorithms are used to speed up the calculations and to reduce both computational cost and energy consumption. Our solution models are simulated using MATLAB (Matrix Laboratory) version R2014b. Accuracy and computational cost are used as performance indicators. Our solution presented remarkable accuracy, with a computational cost four times better than its competitors.

The novelties of the thesis are:

- CORDIC variations are used first time with LOS and SBS NLOS approaches.
- Two new algorithms, ACE-FARM and ACE-DARM, are developed and used in the self-positioning process as they are standing on our unique graphical solution model ACE.
- A unique "Vector Breaking" method is developed and used first time in this study for estimating the position of MO using the arriving signals from the scatters as their locations are unknown.

1.7 Organisation of the Thesis

The thesis is organised as follows. Chapter 2 presents the existing graphical models of solutions, various methods and calculating techniques used for estimating the location of MO in two-dimensional and three-dimensional space. The self-localisation method of Behailu and Tobias (BTS) is decided to be used as a benchmark in comparisons and for appraising the performance of our solutions since it is the closest method to our

graphical model of the solution. In Chapter 3, The BTS is reviewed and its computational cost is calculated. We suggest a new geometric model of solution for self-positioning of MO using LOS and SBS-NLOS signal arriving approaches in Chapter 4. To implement the proposed geometric model of the solution, two new algorithms are developed. These algorithms are presented in Chapter 5. The performance and efficiencies of these algorithms are tested with simulations, simulation results and performance comparisons are discussed in Chapter 6. Finally, Chapter 7 is the conclusion part where the contributions of the study, and what is planned as a future work are explained.

Chapter 2

BACKGROUND

2.1 Introduction

Mobile object (MO) location estimation in 2D and 3D space has become a significant area of research since cellular wireless communication systems and their applications have established a growing place in everyday life. Today, mobile phones have turned into extensions of our hands, so much so that people have become addicted to mobile phones and cannot function without them. Many studies on MO localisation have already been encouraged and supported by the governments, especially for location determination [9, 10]. In 2001, the US Federal Communications Commission (FCC) declared E911 error margin accuracy requirements of 50 meters to 150 meters for handset-based and 100 metres to 300 metres for network-based localisation [9]. In the most recent update (November 2019)[52], the FCC made a vertical (z-axis) accuracy of 3 metres compulsory and claimed that the existing accuracy requirements for the horizontal plane (x-axis and y-axis) would also be tightened and refined by 2023. Various technologies that are vital to public safety utilise location estimation. Electronic bracelets were used first for locating and retrieving lost children and pets. Now they are used to track down people with coronavirus disease 2019 (COVID-19) and those in contact with them [4, 5]. Localisation also plays a special role in fleet management, in that it allows operators (e.g., police forces, taxi and shuttle transportation managers, directors of emergency vehicles) to track and manage their units in an efficient way [6-8]. Location-specific advertisements and marketing are

new trends in the business world. These technologies are used to attract potential customers to products and services that are geographically near them [11-16]. The design, production, and management technologies for unmanned vehicles have been improving at a dizzying speed over the last decade. Some crucial abilities, like self-departure, self-navigation, and self-landing, have been added to the newest generation of drones, allowing them to be operated in every type of condition. The self-navigation ability of a drone requires precise and fast self-positioning techniques that must simultaneously be energy efficient. While the majority of location estimation technologies use satellite Global Positioning Signals (GPS), in dense residential areas and tower blocks, cellular networks' more powerful signals yield better results [1-3].

2.2 Main Methods in Distance and Location Estimation

In the literature, various solutions to the localisation problem are discussed. All of them are built on a model containing base stations (BSs), whose geographic locations are known, and a MO, whose unknown location is estimated using the signals transmitted between the BSs. Many of these methods use fully synchronous clocks to calculate the distances between the components of the model (i.e., the BSs and the MO) via the Time-of-Arrival (TOA) technique [19, 23, 24, 48, 51]. In some articles, the same technique is named as Time-of-Transmission (TOT) or Time-of-Flight (TOF). It calculates the time passed between two time stamps (transmission and arriving times of the signal) and then multiply it with the speed of the signal for calculating the distance. TOA method is presented in Figure 2.1.

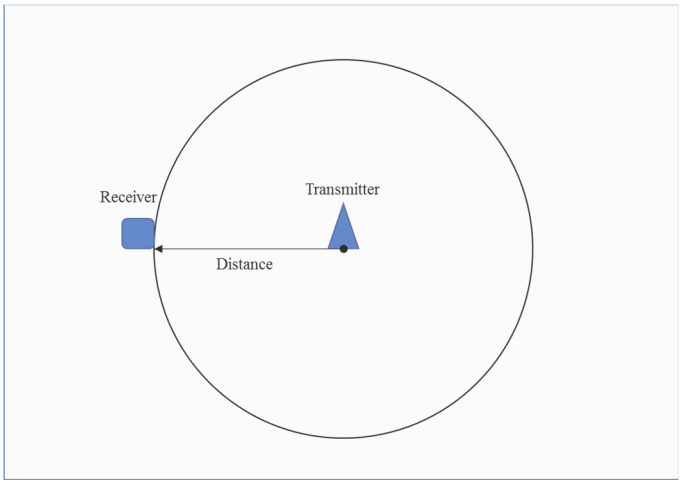

Figure 2.1: Distance calculation with TOA

Time-Difference-of-Arrival (TDOA) is the second popular distance calculation technique and can be used even when the MO clock is not synchronised with those of the BSs [47, 48, 51]. As it is presented in Figure 2.2, in TDOA method, the distance is estimated with the time differences of two signals (TOAs) broadcasted by two transmitters where their locations are known, and their clocks are synchronous with each other but not with a receiver.

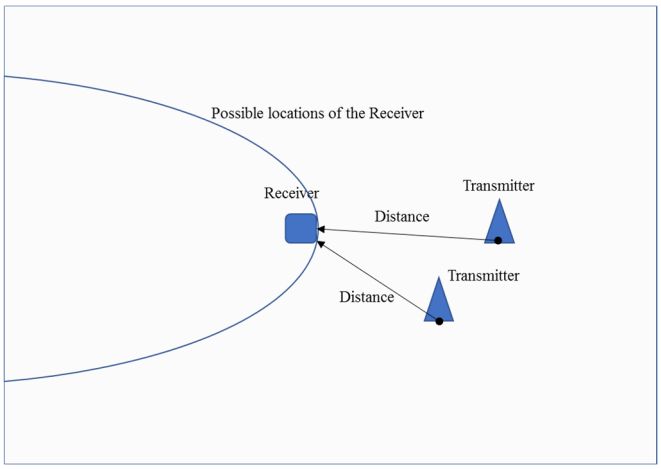

Figure 2.2: Distance estimation with TDOA

Both techniques use total signal transmission time to calculate the distance between the components.

Some of the techniques use the direction of signal propagation, determined with antenna arrays. The Angle-of-Departure (AOD) and Angle-of-Arrival (AOA) values are used in these location estimation processes. The AOD can be calculated using the measured TDOA between the individual elements of the array [48, 49, 51].

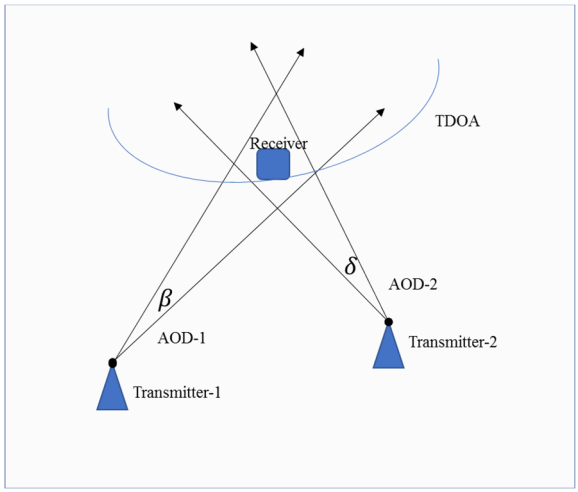

Figure 2.3: Location estimation through AOD and TDOA

Figure 2.3 shows how the receiver estimates its location using AOD and TDOA methods. Detecting and tracking flying objects are essential applications of angle tracking radars, especially in airports. These systems measure the angular changes of the received signals to track their targets [50].

2.3 Signal Arriving Approaches

How the signals are received also plays a huge role in localisation. LOS is one of the most widely acclaimed approaches and is used in many studies [17-21, 42-44] because of its simplicity and relatively low cost of computation. The LOS approach is built on a geometric model, since if there are no obstacles between the BSs and the MO, the sent signals may travel directly to their destinations. The TOA technique with the LOS approach examines the intersection of spheres in 3D (or circles in 2D) space, where the centre points are the known coordinates of the BSs and the radii are the calculated

distances from the MO [47, 51]. As it is shown in Figure 2.4, the common intersection point of all the circles is taken to be the location of the MO in 2D space.

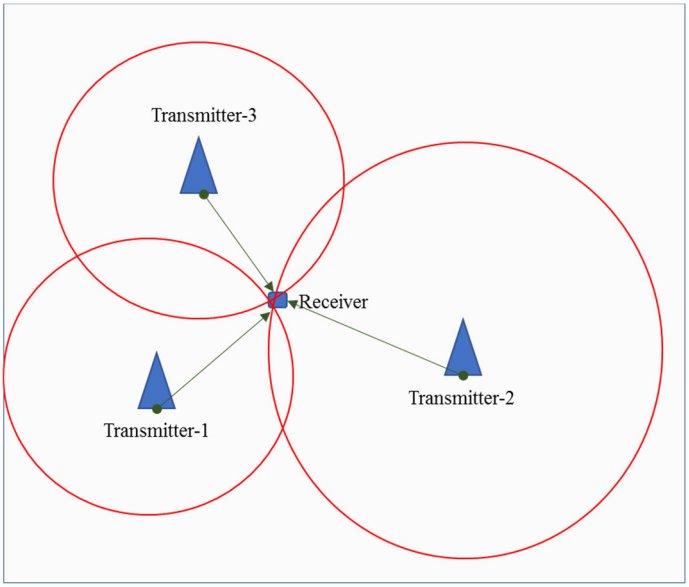

Figure 2.4: Location estimation through LOS and TAO in 2D space

On the other hand, the TDOA LOS technique intersects hyperboloids in the 3D (or hyperboles in 2D) space to locate the MO; the hyperboloids are drawn using the calculated values of the time differences of arriving signals [27, 47, 51]. In practice, Observed Time-Difference-of-Arrival (OTDOA) as presented in Figure 2.5 and Uplink-Time-Difference-of-Arrival (UTDOA) are the two main forms of the TDOA approach. While OTDOA uses the signals transmitted by the BSs and received by the MO to self-position, UTDOA uses the transmitted signals of the MO, which are received by multiple BSs, to estimate the position of the MO on a BS [51].

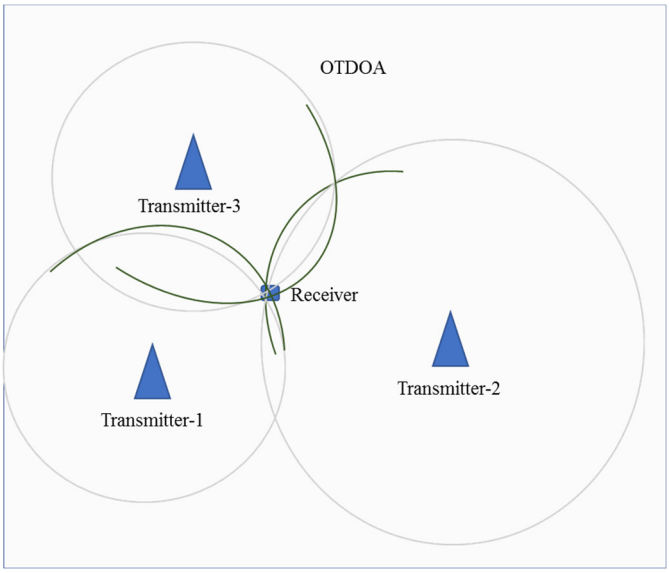

Figure 2.5: Location estimation through LOS, and OTDOA

Cell identification (CID) as presented in Figure 2.6 and enhanced cell identification (eCID) are the basic positioning methods used by GSM providers.

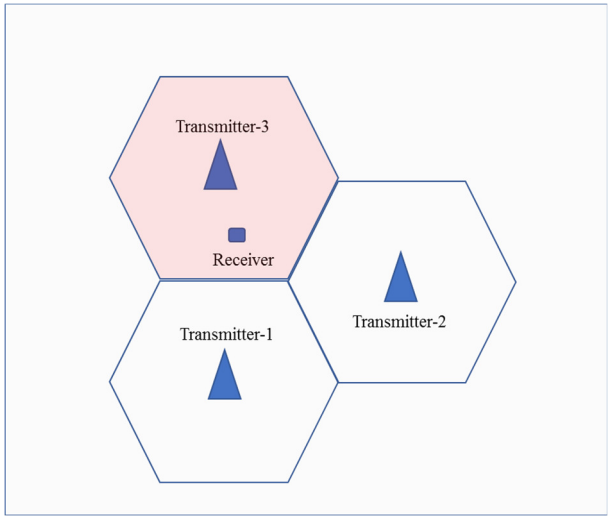

Figure 2.6: Location estimation through CID

These methods detect the serving cell (BS) identity information and its known effective broadcasting range to estimate the possible location of the MO. Additionally, the enhanced version uses the received signals' power and quality along with the Rx-Tx (x^{th} received-transmitted) time difference values of the serving cells to improve

its accuracy [29, 30, 51]. The Received-Signal-Strength (RSS) method is another location estimation method [23, 51] that uses the differences in signal strength between emitted and received signals.

Although LOS is a widely used approach, it encounters difficulties in dense areas and mountainous terrain where direct sight is blocked. In such places, emitted signals may be absorbed by or bounced from scatters (SCs) with unknown locations. The NLOS signal arriving approach aims to estimate the location of an MO under these conditions. Despite its computational complexity, the NLOS approach is more compatible with the multipath nature of the signals, making it more popular in recent studies [22-17 25, 46].

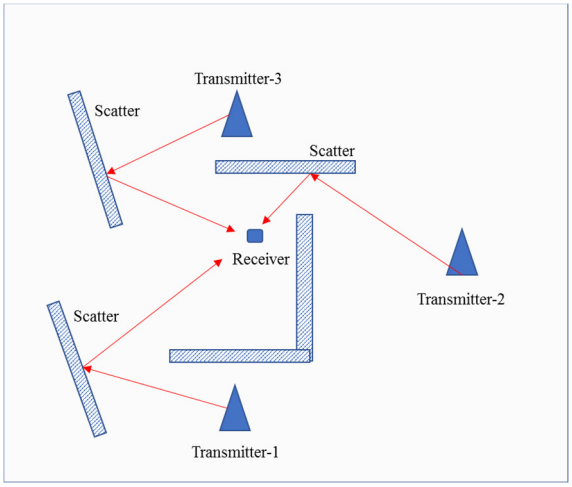

Figure 2.7: Location estimation of MO using NLOS

In the majority of these studies, the location of the MO is calculated on the BSs, and if necessary, this information is sent to the MO, generating an additional networking cost and increasing the time consumed. In only in few studies, the MO can determine its location by using the signals received from the BSs [26, 53, 54]. Figure 2.7 presents how the location of MO can be estimated using NLOS arriving signals of transmitters.

2.4 Calculation Techniques Used in Localisation

Up to this point, the crucial elements of the location estimation process are described as methods and models, but nothing has been told about the algorithms and calculation techniques. Similar geometric models and signal arriving approaches have been used in several solutions and differ from each other only in their calculation techniques. Trigonometric, probabilistic, and statistical calculation techniques [27, 28, 33, 34, 46] are frequently used within localisation models to satisfy the requirements of a variety of applications which each have different accuracy and efficiency requirements in terms of speed and cost of calculation. Many calculation techniques in literature have been reviewed and evaluated thru the metrics: Accuracy, rapidness, and computational cost. It has been noticed that only a few are presenting the desired performance in self-localisation processes of new generation MOs (e.g. smart missiles and unmanned flying devices). This gap in the field motivated us to develop a new self-localisation technique, as it should be applicable for such devices. Hence, in literature, new calculation techniques are researched, as they will improve the accuracy and rapidity, concurrently will reduce the computational costs.

2.4.1 The Coordinate Rotation Digital Computer

The coordinate rotation digital computer (CORDIC) algorithm [35-37, 45] was developed in 1956 and used as a real-time digital resolver on the navigation computers of B-58 bombers to obtain accurate and fast results. It was also used in the navigational systems of the Apollo program's Lunar Roving Vehicle. The technical requirements of CORDIC are very low and it can be executed on any kind of central processing unit (CPU) because it uses simple shift-add operations for several computing tasks. Tasks that can be computed via the CORDIC algorithm include trigonometric, hyperbolic, and logarithmic functions; division; square root calculations; real and complex

multiplication; solutions of linear systems; and eigenvalue estimations. Thus, CORDIC reduces computational costs and makes solutions applicable to both hardware and software by reducing the processor chip area and yielding energy consumption. While emulating heavy weighted mathematical functions, the CORDIC algorithm uses repeated processes for converging to the desired value. Even though it calculates the mathematical functions more rapidly and cost-effectively compared with traditional methods, it has been noticed that the original CORDIC has some performance leakages since it is wasting some time and increasing the cost of computations with some unnecessary repetitions. In the last three decades, the bottlenecks of this algorithm are found and fixed. These studies introduced us to two new CORDIC variations are named "Fixed Angle Rotation Method" and "Dynamic Angle Rotation Method".

2.4.2 Implementation of the Original CORDIC Algorithm

Mathematically, the CORDIC algorithm is represented with a matrix M (see 2.1) and circular rotations with angle α.

$$M = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \tag{2.1}$$

With a sequence of n repetition, $\alpha = \sum_{i=0}^{n} \gamma_i \Delta\varphi_i$, where $\Delta\varphi_i = \arctan(2^{-i})$, and $\gamma_i = \pm 1$ indicating the rotation direction. Hence, the matrix of elementary rotation $M_i = \cos(\Delta\varphi_i) R_i$, and $fM = \prod_{i=0}^{n} R_i$ where $R_i = \begin{pmatrix} 1 & \gamma_i 2^{-1} \\ -\gamma_i 2^{-1} & 1 \end{pmatrix}$ is the rotation matrix at the i[th] iteration. The f is the coordinates lengthening factor as, $f = \prod_{i=0}^{n} \frac{1}{\cos(\Delta\varphi_i)} = \prod_{i=0}^{n} (1 + 2^{-2i})^{\frac{1}{2}}$

Instead of transforming $\begin{pmatrix} x' \\ y' \end{pmatrix} = M \begin{pmatrix} x \\ y \end{pmatrix}$, perform this $f \begin{pmatrix} x' \\ y' \end{pmatrix} = (\prod_{i=0}^{n} R_i) \begin{pmatrix} x \\ y \end{pmatrix}$ then the following algorithm will be used:

$$\begin{pmatrix} x \\ y \end{pmatrix}_{i+1} = \begin{pmatrix} 1 & \gamma_i 2^{-1} \\ -\gamma_i 2^{-1} & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}_i \tag{2.2}$$

$$\varphi_{i+1} = \varphi_i - \gamma_i \Delta\varphi_i, \qquad \Delta\varphi_i = \arctan(2^{-i})$$

$$\gamma_i = sign\, \varphi_i, \qquad i = \overline{0,n}, \qquad \varphi_0 = \alpha$$

where $\begin{pmatrix} x \\ y \end{pmatrix}_0 = \begin{pmatrix} x \\ y \end{pmatrix}$ and $\begin{pmatrix} x \\ y \end{pmatrix}_n = f \begin{pmatrix} x' \\ y' \end{pmatrix}$ as it is shown in (2.2), the algorithm converging to the desired value with just shift and operations which have very small computational costs. (2) is showing the 'rotation' mode of the algorithm.

It is also having the 'vectoring' mode (as in (2.3)).

$$\begin{pmatrix} x \\ y \end{pmatrix}_{i+1} = \begin{pmatrix} 1 & \gamma_i 2^{-1} \\ -\gamma_i 2^{-1} & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}_i, \quad \gamma_i = sign(y_i), \; i = \overline{0,n} \tag{2.3}$$

$$\varphi_{i+1} = \varphi_i - \gamma_i \Delta\varphi_i, \qquad \Delta\varphi_i = \arctan(2^{-i})$$

$$\varphi_0 = 0, \quad \varphi_n \to \alpha, \; y_n \to 0, \; x_n \to f\sqrt{x^2 + y^2}$$

Vectoring mode is containing two parallel operations:

1. Rotating the given 2D vector until it reaches the first canonical axis
2. Calculating the rotation angle

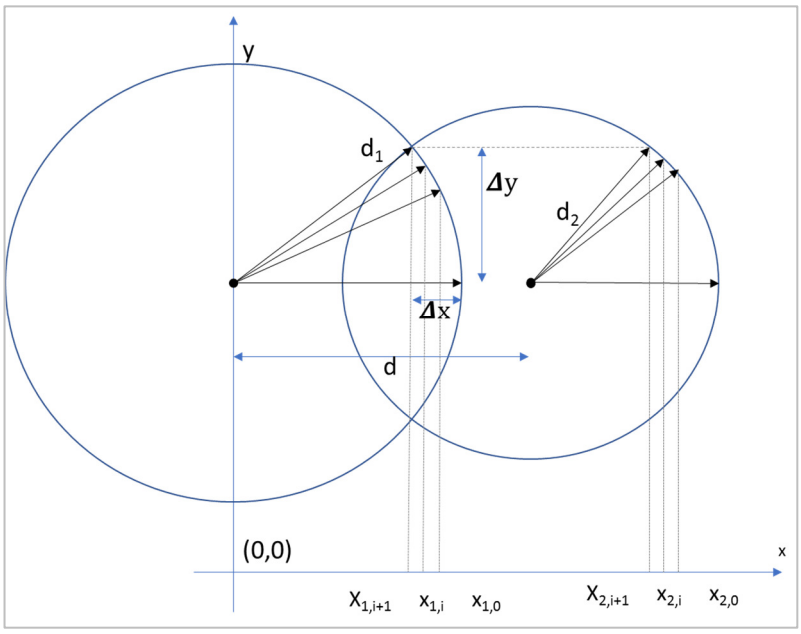

Figure 2.8: Positioning and Vector Rotation in Original CORDIC

Algorithm 2.1: Original CORDIC for converging X_2 to X_1	
Input:	$x_{1,0}$, $x_{2,0}$, $x_{1,0}$, $y_{2,0}$, δ_1 (CORDIC step angle), d_1, d_2
Output	$x_{1,j}$, $x_{2,j}$, $y_{1,i}$, $y_{2,i}$
Begin	
	While $\Delta xj = \lvert x_{1,j} + x_{2,j} - d \rvert > \epsilon$
	Rotate d_1 by one CORDIC step with angle $\delta 1$
	Apply constant factor compensation
	If $\Delta yj = \lvert y_{1,j} - y_{2,j} \rvert > \epsilon$
	Rotate d2 by n CORDIC steps
	Apply constant factor compensation
	End Id
	End While
End	

Rotations are carried until $\Delta y_i = \left| y_2 - y_{1,i} \right| \leq \epsilon$, where y_2 is the targeted value and $y_{1,i}$ is the converging value after the i^{th} iteration. And ϵ is the acceptable error.

Figure 2.8 presents the positioning and rotation processes in the original CORDIC algorithm as its steps are explained in Algorithm 2.1:

In the original CORDIC algorithm, to guarantee the convergence in iterations, the size of the rotation angle δ_i per step (step size) should be always smaller than the increment of y coordinate $\boldsymbol{\Delta}y_{min}$ (see Figure 2.8). Hence, $\boldsymbol{\Delta}y$ should be $\leq \boldsymbol{\Delta}y_{min}$. But, $\boldsymbol{\Delta}y_{min}$ depends on $\boldsymbol{\Delta}x$. The worst-case will be if $d_1 = d = 1$, and $d_2 = \boldsymbol{\Delta}x = 2^{-k}$. where $0 < k \leq 11$. The parameter k is indicating the accepted accuracy level.

2.4.3 Fixed Angle Rotation Method (FARM)

The original CORDIC algorithm always has n iterations in its rotations even with a small angle. Furthermore, it requires compensation for its lengthening factor. They increase the computational cost of the processes. In the FARM variation of CORDIC [42], the original algorithm is modified as:

- Fixed (n-steps) iterations are removed from the algorithm
- The process of calculating the lengthening factor in each step is removed from the algorithm
- Recursive rotations until the desired values are reached added to the algorithm

The stepping angle in radian $\boldsymbol{\delta}_j = \arcsin(2^{-k})$ is used in recursive angle rotations, where k is the parameter letting us decide the desired accuracy level. The approximated sin function is:

$$\sin(\boldsymbol{\delta}_j) = 2^{-k} = v \tag{2.4}$$

where $5 \leq k \leq 11$. The approximated cos function is:

$$\cos(\boldsymbol{\delta}_j) = 1-2^{-(2k+1)} = 1-u \tag{2.5}$$

Then the rotation matrix M will be as follows:

$$M = \begin{pmatrix} \cos(\delta_j) & \sin(\delta_j) \\ -\sin(\delta_j) & \cos(\delta_j) \end{pmatrix} = \begin{pmatrix} 1-2^{-(2k+1)} & 2^{-k} \\ -2^{-k} & 1-2^{-(2k+1)} \end{pmatrix} \tag{2.6}$$

while $\Delta y_i > \epsilon$, vector coordinates (arrowhead) are recursively rotated as follows:

$$x_{j+1} = x_{,j} - x_j\, u + s\, y_{,j}\, v \tag{2.7}$$

$$y_{j+1} = y_{,j} - y_{,j}\, u - s\, x_{,j}\, v \tag{2.8}$$

where s = ±1 the operator of the rotation direction. The $u = 2^{-(2k+1)}$, and $v = 2^{-k}$. As it is seen from the equations (2.7) and (2.8), with just doing simply add, subtract and binary shift operations, heavy weighted trigonometric functions can be emulated. These processes are simplifying the implementations and reducing the computational cost drastically. Algorithm 2.2 is explaining the implementation of the FARM.

Algorithm 2.2: FARM for converging X_2 to X_1

Input: $x_{1,0}$, $x_{2,0}$, $x_{1,0}$, $y_{2,0}$, δ_1 (CORDIC step angle), d_1, d_2
Output $x_{1,j}$, $x_{2,j}$, $y_{1,i}$, $y_{2,i}$
Begin

```
While Δxj = |x1,j + x2,j -d| > ϵ
    Rotate d1 by stepping angle δj
    While Δyj = |y1,j-y2,j| > ϵ
       Rotate d2 by stepping angle δj
    End While
End While
```

End

Figure 2.9 is presenting how sin and cos functions are emulated by FARM. Here the stepping vector is lengthening with one unit (d = 1 unit). Where δ_T is the angle in radian we want to calculate $\sin(\delta_T)$ and $\cos(\delta_T)$. $\delta t(j)$ is the accumulated angle in rotations where it is initialized as zero.

The vector is rotated with stepping angle δ_j on every step where $\delta_{t(j)}$ is indicating the accumulated (total) angle at the j^{th} iteration. The iterations will repeat until the $\delta_{t(j)}$ is getting close enough to δ_T, where their difference is getting less than the acceptable error (ϵ). Then, $\sin(\delta_T) = y_j$ and $\cos(\delta_T) = x_j$

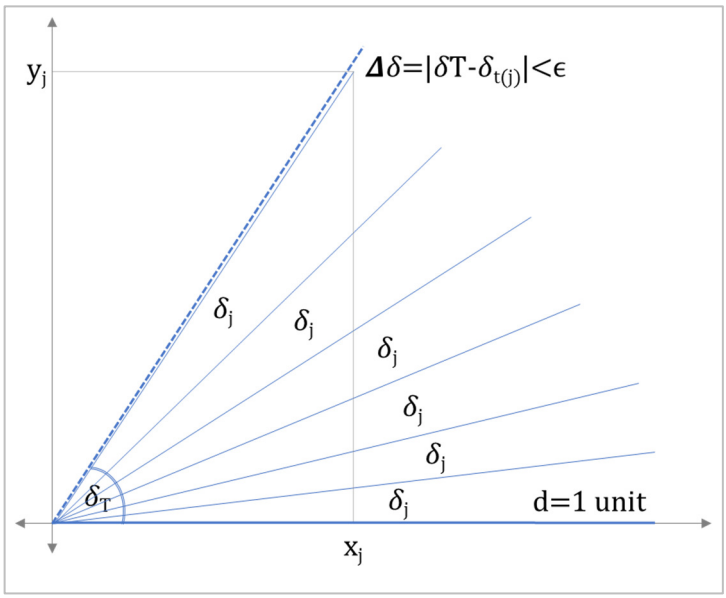

Figure 2.9: Emulating the sin and cos functions with FARM

2.4.4 Dynamic Angle Rotation Method (DARM)

The Fixed Angle Rotation Algorithm has fixed the bottlenecks of the original CORDIC algorithm. FARM is more rapid and computationally cost-effective compared with its ancestor. But, still, there is an opposite relationship between and accuracy and the cost of computations based on the accepted accuracy level (k) where $5 \leq k \leq 11$ [47]. The increase in k improving the accuracy in results exponentially while also increasing the cost of the computations exponentially.

The second variation of CORDIC, "Dynamic Angle Rotation Method" [47] is developed from FARM. The part of the code in the algorithm used for converging to the desired value is modified to improve the accuracy while not increasing the computational cost. In DARM, while converging to the desired value dynamically reducing stepper angle is used instead of a fixed one. Starting with a bigger stepping angle in repetitions letting the algorithm advance faster and it is reducing the computational cost. When it is getting closer to the targeted angle the use of reduced stepping angles allows to get more closer to the desired value in repetitions and it is

improving the accuracy. For calculating the approximated sin function with dynamically changing stepping angle δ_j:

$$v = \sin(\delta j) = \begin{cases} 2^{-k-1} \; if \; \Delta x_j \geq 2^{-k} \\ 2^{-m-1} \; if \; 2^{-m} \leq \Delta x_j < 2^{-m+1} \end{cases} \tag{2.9}$$

Even though the only v is shown in 2.9, u is changing accordingly (see (2.4) and (2.5)). $\boldsymbol{\Delta} x_j = x_{1j} + x_{2j} - x_2$, k=4, m=5, and n=11. Since the distance is normalized, the largest distance will be 1. So, $\boldsymbol{\Delta} x_j < 1$ will be any time.

Algorithm 2.3 shows how the DARM is emulating the sin function. $\delta_{t(j)}$ is the total converged angle at the j^{th} rotation. δ_T is the targeted angle as we want to calculate $\sin(\delta_T)$. $\delta(k)$ is the rotation function that is adjusting the stepping angle δ_k with the accuracy level k. Figure 2.10 illustrates how DARM converging to the aimed angle with dynamically changing stepping angles $\boldsymbol{\delta}_k$, $\boldsymbol{\delta}_{k+1}$, ... where k=4, ..., 11.

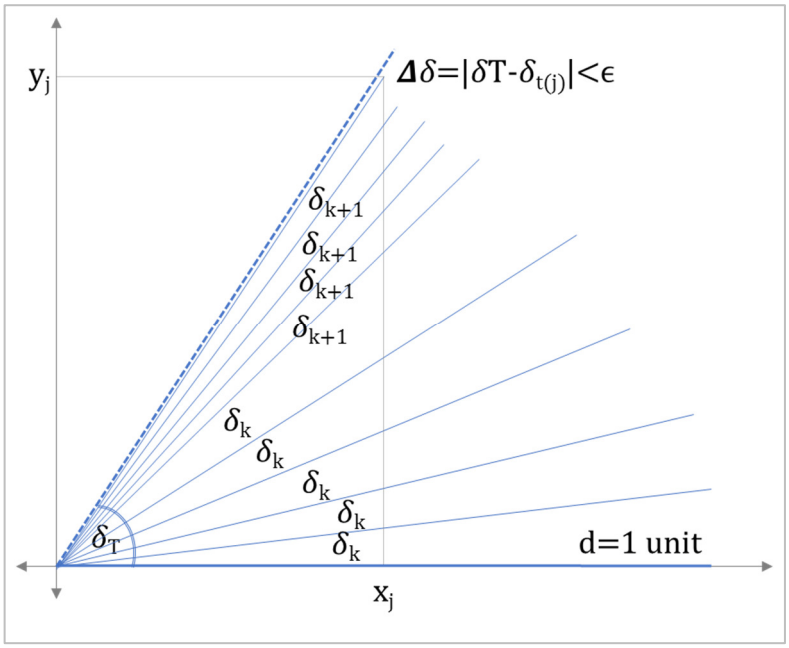

Figure 2.10: Emulating the sin, cos functions with DARM

Algorithm 2.3: DARM for calculating sin(δ_T)

Input: δ_T (targetted angle)
Output y_j
Begin

k = 4; d = 1; $\delta_{t(j)} = 0$
While k ≤ 11
$\delta_k = \delta(k)$; $\epsilon = 2^{-k}$
While $\Delta\delta = |\delta T - \delta_{t(j)}| > \epsilon$
Rotate d_1 by stepping angle δ_k
End While
End While

End

Chapter 3

SURVEY FOR AN ALTERNATIVE SOLUTION

3.1 Introduction

In literature, many approaches for tracking and locating a mobile object in 2D and 3D spaces are reviewed. Then we set the objectives of our proposed system as it has to satisfy the followings:

- It should be applicable to any MO for its self-localisation process and all calculations must be on MO.
- It should be worked not only with LOS but also with NLOS wireless network signals arriving at MO.
- It should be as rapid, accurate and energy conservative enough as it can be used in navigation systems of unmanned flying vehicles and smart missiles.

While developing our model of the solution, concurrently it was researched if there is a study available having a similar objective with us. The main idea of this investigation was to find a benchmark solution for comparing with and apprising our solution.

Since the location estimation technique of Shikur B. and Weber T. (BTS) [46] is having some similar criterion we are looking for, it is decided to be used as a benchmark solution.

The BTS is estimating the location of MO as follows:

- Location estimation is done on MO
- Wireless networking signals broadcasted by base stations are used in the localisation process
- Not only LOS but also SB-NLOS signals are used in the localisation process

The accuracy of BTS is already measured as 34.5 meters (root mean squire error). Since its computational efficiency is not measured in the original study, BTS is reviewed and its computational cost is calculated.

In the rest of this chapter, a review and calculation of the computational cost of BTS are presented.

3.2 Location Estimation Solution of BTS

The founders, Behailu and Tobias (BTS)[46] build their model to be worked with single bounced scattering NLOS arriving signals from BSs to MO. The BS clocks are synchronized with each other but not with MO. The distance of any arriving signal is measured with TDOA. The positioning estimation is done in 3D space with a model containing three BSs, and three scatters. Figure 3.1 shows the single bounced scenario of BTS in 3D space with the l^{th} BS and Scatter pair where aod_{yl}, aod_{zl} are the angle of departures of the signals on X-Y and X-Z planes. In the same way, aoa_{yl}, and aoa_{zl} are the angle of arrivals projections on the X-Y and X-Z planes respectively.

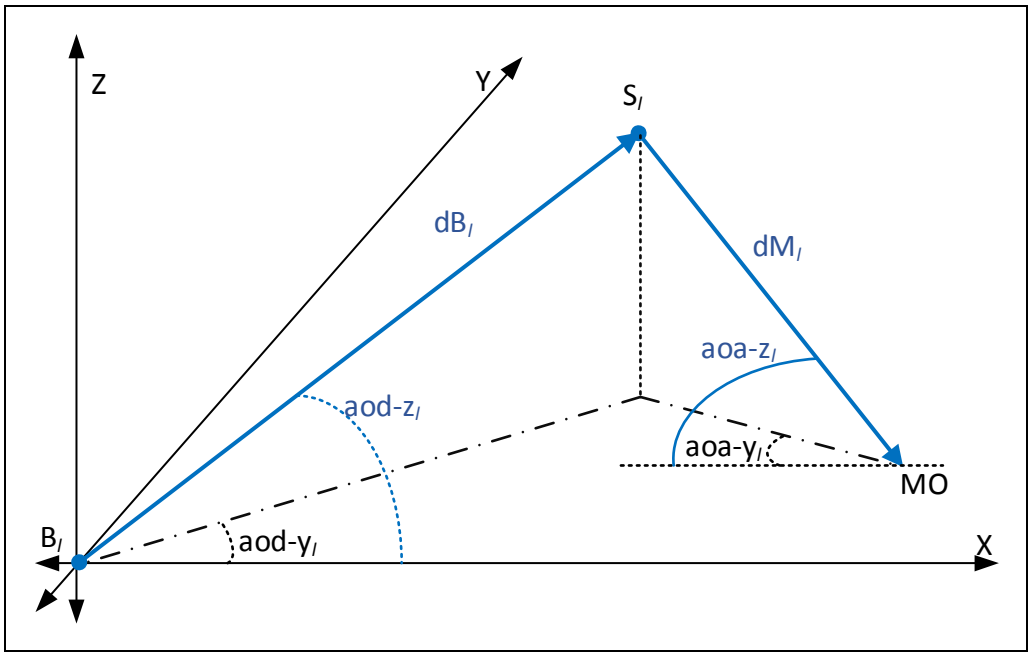

Figure 3.1 System model for one NLOS propagation path in BTS

The D_l is the path length of the l^{th} signal transmitted from BS_l to S_l than to MO. The other components of the model that are demonstrated in Figure 3.1 are calculated as follows:

$$d_l = d_{Bl} + d_{Ml} \tag{3.1}$$

$$d_{Bl} = \sqrt{(x_{Sl} - x_{Bl})^2 + (y_{Sl} - y_{Bl})^2 + (z_{Sl} - z_{Bl})^2} \tag{3.2}$$

$$d_{Ml} = \sqrt{(x_{Sl} - x_M)^2 + (y_{Sl} - y_M)^2 + (z_{Sl} - z_M)^2} \tag{3.3}$$

$$\Delta d_l = d_l - d_1 \tag{3.4}$$

$$aod_{yl} = \frac{\pi}{2}(1 - sgn(x_{sl} - x_{Bl}) + tan^{-1}\frac{y_{Sl} - y_{Bl}}{x_{Sl} - x_{Bl}} \tag{3.5}$$

$$aoa_{yl} = \frac{\pi}{2}(1 - sgn(x_{sl} - x_M) + tan^{-1}\frac{y_{sl} - y_M}{x_{Sl} - x_M} \tag{3.6}$$

$$aod_{zl} = \frac{\pi}{2} - tan^{-1}\frac{z_{Sl} - z_{Bl}}{\sqrt{(x_{Sl} - x_{Bl})^2 + (y_{Sl} - y_{Bl})^2}} \tag{3.7}$$

$$aoa_{zl} = \frac{\pi}{2} - tan^{-1}\frac{z_{Sl} - z_M}{\sqrt{(x_{Sl} - x_M)^2 + (y_{Sl} - y_M)^2}} \tag{3.8}$$

The calculated length of the path is d_l, $\boldsymbol{\Delta}d_l$ is the path length difference, and Θ is the state vector that is containing the coordinates of the scatters and MO.

$$m = h(\theta) + w \tag{3.9}$$

BTS considers the possible noisy estimates on $\boldsymbol{\Delta}d_l$, aods and aoas in noisy observation vector *m* as shown on (3.9) where $h(\Theta)$ is observation vector function. The errors in estimations caused by noise in observations *(w)* are reduced reasonably with the MUSIC algorithm [39].

3.2.1 Formulation of the Geometric Solution

In this section, the formulation of location estimation processes used by BTS is explained. This model uses *l*, actually used 4, base station – scatter pairs and one MO as:

$$x_{Sl} = x_{Bl} + d_{Bl}\sin(aod_{zl})\cos(aod_{yl}) \tag{3.10}$$

$$y_{Sl} = y_{Bl} + d_{Bl}\sin(aod_{Zl})\sin(aod_{Yl}) \tag{3.11}$$

$$z_{Sl} = z_{Bl} + d_{Bl}\cos(aod_{Zl}) \tag{3.12}$$

$$\begin{aligned} x_M &= x_{Sl} - (d_l - d_{Bl})\sin(aoa_{Zl})\cos(aoa_{Yl}) \\ &= x_{Bl} + d_{Bl}a_{l,1} - (d_1 + \Delta d_l)\sin(aoa_{Zl})\cos(aoa_{Yl}) \end{aligned} \tag{3.13}$$

$$\begin{aligned} y_M &= y_{Sl} - (d_l - d_{Bl})\sin(aoa_{Zl})\sin(aoa_{Yl}) \\ &= y_{Bl} + d_{Bl}a_{l,2} - (d_1 + \Delta d_l)\sin(aoa_{Zl})\sin(aoa_{Yl}) \end{aligned} \tag{3.14}$$

$$\begin{aligned} z_M &= z_{Sl} - (d_l - d_{Bl})\cos(aoa_{Zl}) \\ &= z_{Bl} + d_{Bl}a_{l,3} - (d_1 + \Delta d_l)\cos(aoa_{Zl}) \end{aligned} \tag{3.15}$$

where

$$a_{l,1} = \sin(aod_{Zl})\cos(aod_{Yl}) + \sin(aoa_{Zl})\cos(aoa_{Yl}) \tag{3.16}$$

$$a_{l,2} = \sin(aod_{Zl})\sin(aod_{Yl}) + \sin(aoa_{Zl})\sin(aoa_{Yl}) \tag{3.17}$$

$$a_{l,3} = \cos(aod_{Zl}) + \cos(aoa_{Zl}) \tag{3.18}$$

From (3.13) rewrite the d_{Bl} *as*

$$d_{Bl} = \frac{x_M - x_{Bl} + (d_1 + \Delta d_l)\sin(aoa_{Zl})\cos(aoa_{Yl})}{a_{l,1}} \quad (3.19)$$

Substituting (3.14) and (3.15) with (3.19),

$$-x_M a_{l,2} + y_M a_{l,1} = -x_{Bl} a_{l,2} + y_{Bl} a_{l,1} + (d_1 + \Delta d_l) a_{l,4} \quad (3.20)$$

$$-x_M a_{l,3} + z_M a_{l,1} = -x_{Bl} a_{l,3} + z_{Bl} a_{l,1} + (d_1 + \Delta d_{l)} a_{l,5} \quad (3.21)$$

where

$$a_{l,4} = \sin(aoa_{Zl})\sin(aod_{Zl})\sin(aod_{Yl}) - aoa_{Yl} \quad (3.22)$$

$$a_{l,5} = \sin(aoa_{Zl})\cos(aoa_{Yl})\cos(aod_{Zl}) - \cos(aoa_{Zl})\sin(aod_{Zl})\cos(aod_{Yl}) \quad (3.23)$$

For each BS-scatter pair (l = 1, 2, …) similar equations are generated and a matrix A is defined as follows:

$$A = (a_1, a_2, \ldots)^T \quad (3.24)$$

where

$$a_{2l-1} = \left(-a_{l,2},\ a_{l,1},\ 0,\ -a_{l,4}\right)^T \quad (3.25)$$

$$a_{2l} = \left(-a_{l,3},\ \ 0,\ \ \ a_{l,1},\ -a_{l,5}\right)^T \quad (3.26)$$

The unknowns vector r

$$r = (x_M,\ y_M,\ x_M,\ d_1)^T \quad (3.27)$$

where d1 is a nuisance parameter and the vector b indicate then knowns

$$b = (b_1,\ b_2, \ldots)^T \quad (3.26)$$

$$b_{2l-1} = -x_{Bl} a_{l,2} + y_{Bl} a_{l,1} + \Delta d_l a_{l,4} \quad (3.27)$$

$$b_{2l} = -x_{Bl} a_{l,3} + z_{Bl} a_{l,1} + \Delta d_l a_{l,5} \quad (3.28)$$

Finally, the system of linear equation is constructed from (3.20) and (3.21) using A, r and b as

$$Ar = b \quad (3.29)$$

Using the pseudo-inverse of the matrix A the unknowns r can be written as

$$r = (A^T A)^{-1} A^T b \tag{3.30}$$

3.2.2 Computational Cost of BTS

Table 3.1: Weight of operations

Operation	**Addition**	**Subtraction**	**Shift**	**Multiplication**
Weight	1	1	1	40
Operation	**Division**	**Sin**	**Cos**	**Tan**
Weight	40	404	404	1448

In Section 3.1.1, formulation of the BTS was described and the location estimation process of that model is represented with a linear equation shown at (3.29). In this section, the computational cost of BTS will be calculated to be used as a benchmark for the rest of the study. All processes are spending the processor cycles since the performance of the methods and algorithms are measure with their computational costs.

In Table 3.1, the computational costs of the arithmetic and trigonometric operations are listed [13]. On the way of calculating the vector of unknowns *r*, first the known vectors *A* and *b* must be defined and placed to the proper locations of the equation.

The known values of the model are presented in Table 3.2. Similar to our approach, the coordinates of BSs in the 3D space, angles of departures on BSs and angles of arrivals on MO are known values on the BTS's model.

Table 3.2: The known values of the BTS's model

Value	**Explanation**
x_{B1}	Place of BS_1 on X-axis
y_{B1}	Place of BS_1 on Y-axis
z_{B1}	Place of BS_1 on Z-axis
$\alpha_1 = aod_{Z1}$	The angle of departure from BS_1 on X-Z axis
$\psi_1 = aod_{Y1}$	The angle of departure from BS_1 on X-Y axis
$\beta_1: aoa_{Z1}$	The angle of arrival from SC_1 on X-Z axis
$\phi_1: aoa_{Y1}$	The angle of arrival from SC_1 on X-Y axis
x_{B2}	Place of BS_2 on X-axis
y_{B2}	Place of BS_2 on Y-axis
z_{B2}	Place of BS_2 on Z-axis
$\alpha_2: aod_{Z2}$	The angle of departure from BS_2 on X-Z axis
$\psi_2: aod_{Y2}$	The angle of departure from BS_2 on X-Y axis
$\beta_2: aoa_{Z2}$	The angle of arrival from SC_2 on X-Z axis,
$\phi_2: aoa_{Y2}$	The angle of arrival from SC_2 on X-Y axis
x_{B3}	Place of BS_3 place on X-axis
y_{B3}	Place of BS_3 place on Y-axis
z_{B3}	Place of BS_3 place on Z-axis
$\alpha_3: aod_{Z3}$	The angle of departure from BS_3 on X-Z axis
$\psi_3: aod_{Y3}$	The angle of departure from BS_3 on X-Y axis
$\beta 3: aoa_{Z3}$	The angle of arrival from SC_3 on X-Z axis
$\phi 3: aoa_{Y3}$	The angle of arrival from SC3 on X-Y axis

The geometric solution as described in section 3.1.1 is based on the information generated by three base stations. So, the vectors A, b and r of the linear equation (3.29) should be re-written as in (3.31);

$$A = \begin{bmatrix} -a_{1,2} & a_{1,1} & 0 & -a_{1,4} \\ -a_{1,3} & 0 & a_{1,1} & -a_{1,5} \\ -a_{2,2} & a_{2,1} & 0 & -a_{2,4} \\ -a_{2,3} & 0 & a_{2,1} & -a_{2,5} \\ -a_{3,2} & a_{3,1} & 0 & -a_{3,4} \\ -a_{3,3} & 0 & a_{3,1} & -a_{3,5} \end{bmatrix} \qquad r = \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ d_1 \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix} \tag{3.31}$$

where d_1 is the nuisance parameter here.

3.2.3 The Computational Costs of Vectors A and b

For finding the computational costs (CCost) of vectors A and b, the costs of their parameters are calculated first. Then, these partial costs are summed for calculating the computational costs of the vectors. (see Table 3.3 and Table 3.4). The vector r is containing the estimated coordinates of the MO in the 3D space. Hence, the parameters of vector r are the unknowns of this linear system. For calculating these unknown parameters, the vector r in (3.29) has to be kept on the left side of the equation alone. Therefore both sides of the equation must be multiplied with the inverse of vector A as described in (3.32) and the resulting equation will be as in (3.33).

$$(A^{-1}A)\, r = A^{-1}\, b \tag{3.32}$$

$$r = A^{-1}\, b \tag{3.33}$$

Since A is not a square matrix, we cannot get its inverse directly and we need to use pseudo-inverse of matrix A as in (3.34)

$$r = (A^{T}A)^{-1}\, A^{T}\, b \tag{3.34}$$

Table 3.3: The computational cost calculation of vector A.

Parameters of vector A	**CCost**
$a_{1,1} = \sin \alpha_1 \cos \psi_1 + \sin \beta_1 \cos \phi_1$	1697
$a_{1,2} = \sin \alpha_1 \sin \psi_1 + \sin \beta_1 \sin \phi_1$	1697
$a_{1,3} = \cos \alpha_1 + \cos \beta_1$	809
$a_{1,4} = \sin \alpha_1 \sin \beta_1 \sin \psi_1 - \phi_1$	1293
$a_{1,5} = \sin \beta_1 \cos \phi_1 \cos \alpha_1 - \cos \beta_1 \sin \alpha_1 \cos \psi_1$	2585
$a_{2,1} = \sin \alpha_2 \cos \psi_2 + \sin \beta_2 \cos \phi_2$	1697
$a_{2,2} = \sin \alpha_2 \sin \psi_2 + \sin \beta_2 \sin \phi_2$	1697
$a_{2,3} = \cos \alpha_2 + \cos \beta_2$	809
$a_{2,4} = \sin \alpha_2 \sin \beta_2 \sin \psi_2 - \phi_2$	1293
$a_{2,5} = \sin \beta_2 \cos \phi_2 \cos \alpha_2 - \cos \beta_2 \sin \alpha_2 \cos \psi_2$	2585
$a_{3,1} = \sin \alpha_3 \cos \psi_3 + \sin \beta_3 \cos \phi_3$	1697
$a_{3,2} = \sin \alpha_3 \sin \psi_3 + \sin \beta_3 \sin \phi_3$	1697
$a_{3,3} = \cos \alpha_3 + \cos \beta_3$	809
$a_{3,4} = \sin \alpha_3 \sin \beta_3 \sin \psi_3 - \phi_3$	1293
$a_{3,5} = \sin \beta_3 \cos \phi_3 \cos \alpha_3 - \cos \beta_3 \sin \alpha_3 \cos \psi_3$	2585
The total computational cost of vector A	24243

Table 3.4: The computational cost calculation of vector b.

Parameters of vector b	**CCost**
$b_1 = -x_{B1} a_{1,2} + y_{B1} a_{1,1} + \delta_1 a_{1,4}$	123
$b_2 = -x_{B1} a_{1,3} + z_{B1} a_{1,1} + \delta_1 a_{1,5}$	123
$b_3 = -x_{B2} a_{2,2} + y_{B2} a_{2,1} + \delta_2 a_{2,4}$	123
$b_4 = -x_{B2} a_{2,3} + z_{B2} a_{2,1} + \delta_2 a_{2,5}$	123

$b_5 = -x_{B3}\, a_{3,2} + y_{B3}\, a_{3,1} + \delta_3\, a_{3,4}$	123
$b_6 = -x_{B3}\, a_{3,3} + z_{B3}\, a_{3,1} + \delta_3\, a_{3,5}$	123
The total computational cost of vector A	738

Thereafter, vector A is simplified as in (3.35).

$$A = \begin{bmatrix} -a_{1,2} & a_{1,1} & 0 & -a_{1,4} \\ -a_{1,3} & 0 & a_{1,1} & -a_{1,5} \\ -a_{2,2} & a_{2,1} & 0 & -a_{2,4} \\ -a_{2,3} & 0 & a_{2,1} & -a_{2,5} \\ -a_{3,2} & a_{3,1} & 0 & -a_{3,4} \\ -a_{3,3} & 0 & a_{3,1} & -a_{3,5} \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 & c_4 \\ c_5 & c_6 & c_7 & c_8 \\ c_9 & c_{10} & c_{11} & c_{12} \\ c_{13} & c_{14} & c_{15} & c_{16} \\ c_{17} & c_{18} & c_{19} & c_{20} \\ c_{21} & c_{22} & c_{23} & c_{24} \end{bmatrix} \quad (3.35)$$

$$\text{Let say } E = (A^T A) = \begin{bmatrix} d_1 & d_2 & d_3 & d_4 \\ d_5 & d_6 & d_7 & d_8 \\ d_9 & d_{10} & d_{11} & d_{12} \\ d_{13} & d_{14} & d_{15} & d_{16} \end{bmatrix}$$

Table 3.5: The computational cost calculation of E

Parameters of vector E	**CCost**
$d_1 = c_1\, c_1 + c_5\, c_5 + c_9\, c_9 + c_{13}\, c_{13} + c_{17}\, c_{17} + c_{21}\, c_{21}$	245
$d_2 = c_1\, c_2 + c_5\, c_6 + c_9\, c_{10} + c_{13}\, c_{14} + c_{17}\, c_{18} + c_{21}\, c_{22}$	245
$d_3 = c_1\, c_1 + c_5\, c_7 + c_9\, c_{11} + c_{13}\, c_{15} + c_{17}\, c_{19} + c_{21}\, c_{23}$	245
$d_4 = c_1\, c_1 + c_5\, c_8 + c_9\, c_{12} + c_{13}\, c_{16} + c_{17}\, c_{20} + c_{21}\, c_{24}$	245
$d_5 = c_2\, c_1 + c_6\, c_5 + c_{10}\, c_9 + c_{14}\, c_{13} + c_{18}\, c_{17} + c_{22}\, c_{21}$	245
$d_6 = c_2\, c_2 + c_6\, c_6 + c_{10}\, c_{10} + c_{14}\, c_{14} + c_{18}\, c_{18} + c_{22}\, c_{22}$	245
$d_7 = c_2\, c_1 + c_6\, c_7 + c_{10}\, c_{11} + c_{14}\, c_{15} + c_{18}\, c_{19} + c_{22}\, c_{23}$	245
$d_8 = c_2\, c_1 + c_6\, c_8 + c_{10}\ c_{12} + c_{14}\, c_{16} + c_{18}\, c_{20} + c_{22}\, c_{24}$	245
$d_9 = c_3\, c_1 + c_7\, c_5 + c_{11}\, c_9 + c_{15}\, c_{13} + c_{19}\, c_{17} + c_{23}\, c_{21}$	245
$d_{10} = c_3\, c_2 + c_7\, c_6 + c_{11}\, c_{10} + c_{15}\, c_{14} + c_{19}\, c_{18} + c_{23}\, c_{22}$	245
$d_{11} = c_3\, c_1 + c_7\, c_7 + c_{11}\, c_{11} + c_{15}\, c_{15} + c_{19}\, c_{19} + c_{23}\, c_{23}$	245

$d_{12} = c_3 c_1 + c_7 c_8 + c_{11} c_{12} + c_{15} c_{16} + c_{19} c_{20} + c_{23} c_{24}$	245
$d_{13} = c_4 c_1 + c_8 c_5 + c_{12} c_9 + c_{16} c_{13} + c_{20} c_{17} + c_{24} c_{21}$	245
$d_{14} = c_4 c_2 + c_8 c_6 + c_{12} c_{10} + c_{16} c_{14} + c_{20} c_{18} + c_{24} c_{22}$	245
$d_{15} = c_4 c_1 + c_8 c_7 + c_{12} c_{11} + c_{16} c_{15} + c_{20} c_{19} + c_{24} c_{23}$	245
$d_{16} = c_4 c_1 + c_8 c_8 + c_{12} c_{12} + c_{16} c_{16} + c_{20} c_{20} + c_{24} c_{24}$	245
The total computational cost of E	3920

Where parameters of E and their computational costs (CCost) are calculated and described in Table 3.5. Then $(A^TA)^{-1} = E^{-1} = \frac{1}{\det(E)}\ adj(E)$ is calculated, where steps of processes and their computational costs (CCost) are presented in Table 3.6.

Table 3.6: The computational cost calculation of E^{-1}

Processes	**CCost**
$\det(E) = d_1 \det(E_1) - d_2 \det(E_2) + d_3 \det(E_3) - d_4 \det(E_4)$	163
$\det(E_1) = d_6 \det(E_{1,1}) - d_7 \det(E_{1,2}) + d_8 \det(E_{1,3})$	123
$\det(E_{1,1}) = d_{11} d_{16} - d_{12} d_{15}$	81
$\det(E_{1,2}) = d_{10} d_{16} - d_{12} d_{14}$	81
$\det(E_{1,3}) = d_{10} d_{15} - d_{11} d_{15}$	81
$\det(E2) = d_5 \det(E_{2,1}) - d_7 \det(E_{2,2}) + d_8 \det(E_{2,3})$	123
$\det(E_{2,1}) = d_{11} d_{16} - d_{12} d_{15}$	81
$\det(E_{2,2}) = d_{10} d_{16} - d_{12} d_{14}$	81
$\det(E_{2,3}) = d_{10} d_{15} - d_{11} d_{15}$	81
$\det(E3) = d_6 \det(E_{3,1}) - d_7 \det(E_{3,2}) + d_8 \det(E_{3,3})$	123
$\det(E_{3,1}) = d_{11} d_{16} - d_{12} d_{15}$	81

$\det(E_{3,2}) = d_{10}\, d_{16} - d_{12}\, d_{14}$	81
$\det(E_{3,3}) = d_{10}\, d_{15} - d_{11}\, d_{15}$	81
$\det(E_4) = d_6 \det(E_{4,1}) - d_7 \det(E_{4,2}) + d_8 \det(E_{4,3})$	123
$\det(E_{4,1}) = d_{11}\, d_{16} - d_{12}\, d_{15}$	81
$\det(E_{4,2}) = d_{10}\, d_{16} - d_{12}\, d_{14}$	81
$\det(E_{4,3}) = d_{10}\, d_{15} - d_{11}\, d_{15}$	81
The total computational cost of E^{-1}	1627

On the next step, let adj(E) =C^T, so C has to be as in (3.36)

$$C=\begin{bmatrix} +\det(F_1) & -\det(F_2) & +\det(F_3) & -\det(F_4) \\ -\det(F_5) & +\det(F_6) & -\det(F_7) & +\det(F_8) \\ +\det(F_9) & -\det(F_{10}) & +\det(F_{11}) & -\det(F_{12}) \\ -\det(F_{13}) & +\det(F_{14}) & -\det(F_{15}) & +\det(F_{16}) \end{bmatrix} \tag{3.36}$$

The elements of C in (3.36) are written as:

$$+\det(F_1)=+\begin{vmatrix} d_6 & d_7 & d_8 \\ d_{10} & d_{11} & d_{12} \\ d_{14} & d_{15} & d_{16} \end{vmatrix}, \quad -\det(F_2)=-\begin{vmatrix} d_5 & d_7 & d_8 \\ d_9 & d_{11} & d_{12} \\ d_{13} & d_{15} & d_{16} \end{vmatrix},$$

$$+\det(F_3)=+\begin{vmatrix} d_5 & d_6 & d_8 \\ d_9 & d_{10} & d_{12} \\ d_{13} & d_{14} & d_{16} \end{vmatrix}, \quad -\det(F_4)=-\begin{vmatrix} d_5 & d_6 & d_7 \\ d_9 & d_{10} & d_{11} \\ d_{13} & d_{14} & d_{12} \end{vmatrix},$$

$$+\det(F_5)=+\begin{vmatrix} d_2 & d_3 & d_4 \\ d_{10} & d_{11} & d_{12} \\ d_{14} & d_{15} & d_{16} \end{vmatrix}, \quad -\det(F_6)=-\begin{vmatrix} d_1 & d_3 & d_4 \\ d_9 & d_{11} & d_{12} \\ d_{13} & d_{15} & d_{16} \end{vmatrix},$$

$$+\det(F_7)=+\begin{vmatrix} d_1 & d_2 & d_4 \\ d_9 & d_{10} & d_{12} \\ d_{13} & d_{14} & d_{16} \end{vmatrix}, \quad -\det(F_8)=-\begin{vmatrix} d_1 & d_2 & d_3 \\ d_9 & d_{10} & d_{11} \\ d_{13} & d_{14} & d_{12} \end{vmatrix},$$

$$+\det(F_9)=+\begin{vmatrix} d_2 & d_3 & d_4 \\ d_6 & d_7 & d_8 \\ d_{14} & d_{15} & d_{16} \end{vmatrix}, \quad -\det(F_{10})=-\begin{vmatrix} d_1 & d_3 & d_4 \\ d_5 & d_7 & d_8 \\ d_{13} & d_{15} & d_{16} \end{vmatrix},$$

$$+\det(F_{11})=+\begin{vmatrix} d_1 & d_2 & d_4 \\ d_5 & d_6 & d_8 \\ d_{13} & d_{14} & d_{16} \end{vmatrix}, \quad -\det(F_{12})=-\begin{vmatrix} d_1 & d_2 & d_3 \\ d_5 & d_6 & d_7 \\ d_{13} & d_{14} & d_{15} \end{vmatrix},$$

$$+\det(F_{13}) = +\begin{vmatrix} d_2 & d_3 & d_4 \\ d_6 & d_7 & d_8 \\ d_{10} & d_{11} & d_{12} \end{vmatrix}, \quad -\det(F_{14}) = -\begin{vmatrix} d_1 & d_3 & d_4 \\ d_5 & d_7 & d_8 \\ d_9 & d_{11} & d_{12} \end{vmatrix},$$

$$+\det(F_{15}) = +\begin{vmatrix} d_1 & d_2 & d_4 \\ d_5 & d_6 & d_8 \\ d_9 & d_{10} & d_{12} \end{vmatrix}, \quad -\det(F_{16}) = -\begin{vmatrix} d_1 & d_2 & d_3 \\ d_5 & d_6 & d_7 \\ d_9 & d_{10} & d_{11} \end{vmatrix}$$

Then, the computational cost of each element is calculated. The processes and the calculated costs are presented in Table 3.7, and Table 3.8.

Table 3.7: The computational cost calculation of C – Part 1

Processes	**CCost**
$+\det(F_1) = +d_6 (d_{11}d_{16} - d_{12} d_{15}) - d_7 (d_{10} d_{16} - d_{14} d_{12}) + d_8 (d_{10} d_{15} - d_{13} d_{10})$ $= +d_6 d_{11} d_{16} - d_6 d_{12} d_{15} - d_7 d_{10} d_{16}$ $+d_7 d_{14} d_{12} + d_8 d_{10} d_{15} - d_8 d_{13} d_{10}$	486
$-\det(F_2) = -[d_5 (d_{11} d_{16} - d_{12} d_{15}) - d_7 (d_9 d_{16} - d_{12} d_{13}) + d_8 (d_9 d_{15} - d_{11} d_{13})]$ $= - d_5 d_{11} d_{16} + d_5 d_{12} d_{15} + d_7 d_9 d_{16}$ $- d_7 d_{12} d_{13} - d_8 d_9 d_{15} + d_8 d_{11} d_{13}$	486
$+\det(F_3) = +d_5 (d_{10} d_{16} - d_{12} d_{14}) - d_6 (d_9 d_{16} - d_{12} d_{13}) + d_8 (d_9 d_{14} - d_{10} d_{13})$ $=+d_5 d_{10} d_{16} - d_5 d_{12} d_{14} - d_6 d_9 d_{16}$ $+d_6 d_{12} d_{13} + d_8 d_9 d_{14} - d_8 d_{10} d_{13}$	486
$-\det(F_4) = -[d_5 (d_{10} d_{12} - d_{11} d_{14}) - d_6 (d_9 d_{12} - d_{11} d_{13}) + d_7 (d_9 d_{14} - d_{10} d_{13})]$ $= -d_5 d_{10} d_{12} + d_5 d_{11} d_{14} + d_6 d_9 d_{12}$ $+d_6 d_{11} d_{13} + d_7 d_9 d_{14} - d_7 d_{10} d_{13}$	486
$+\det(F_5) =+d_2 (d_{11} d_{16} - d_{12} d_{15}) - d_3 (d_{10} d_{16} - d_{12} d_{14}) + d_4 (d_{10} d_{15} - d_{11} d_{14})$ $=+d_2 d_{11} d_{16} - d_2 d_{12} d_{15} - d_3 d_{10} d_{16}$ $+d_3 d_{12} d_{14} + d_4 d_{10} d_{15} - d_4 d_{11} d_{14}$	486
$-\det(F_6)= -[d_1 (d_{11} d_{16} - d_{12} d_{15}) - d_3 (d_9 d_{16} - d_{12} d_{13}) + d_4 (d_9 d_{15} - d_{11} d_{13})]$ $= -d_1 d_{11} d_{16} + d_1 d_{12} d_{15} + d_3 d_9 d_{16}$	

$-d_3\, d_{12}\, d_{13} - d_4\, d_9\, d_{15} + d_4\, d_{11}\, d_{13}$	486
$+\det(F_7) = +d_1\,(d_{10}\, d_{16} - d_{12}\, d_{14}) - d_2\,(d_9\, d_{16} - d_{12}\, d_{13}) + d_4\,(d_9\, d_{14} - d_{10}\, d_{13})$ $= +d_1\, d_{10}\, d_{16} - d_1\, d_{12}\, d_{14} - d_2\, d_9\, d_{16}$ $+d_2\, d_{12}\, d_{13} + d_4\, d_9\, d_{14} - d_4\, d_{10}\, d_{13}$	486
$-\det(F_8) = -[d_1\,(d_{10}\, d_{12} - d_{11}\, d_{14}) - d_2\,(d_9\, d_{12} - d_{11}\, d_{13}) + d_3\,(d_9\, d_{14} - d_{10}\, d_{13})]$ $= -d_1\, d_{10}\, d_{12} + d_1\, d_{11}\, d_{14} + d_2\, d_9\, d_{12}$ $-d_2\, d_{11}\, d_{13} - d_3\, d_9\, d_{14} + d_3\, d_{10}\, d_{13}$	486
The total computational cost of Part 1	3888

Table 3.8: The computational cost calculation of C – Part 2

Processes	**CCost**
$+\det(F_9) = +d_2\,(d_7\, d_{16} - d_8\, d_{15}) - d_3(d_6\, d_{16} - d_8\, d_{14}) + d_4\,(d_6\, d_{15} - d_7\, d_{14})$ $= +d_2\, d_7\, d_{16} - d_2\, d_8\, d_{15} - d_3\, d_6\, d_{16}$ $+d_3\, d_8\, d_{14} + d_4\, d_6\, d_{15} - d_4\, d_7\, d_{14}$	486
$-\det(F_{10}) = -[d_1\,(d_7\, d_{15} - d_8\, d_{15}) - d_3\,(d_5\, d_{16} - d_8 d_{13}) + d_4\,(d_5\, d_{15} - d_7\, d_{13})]$ $= -d_1\, d_7\, d_{15} + d_1\, d_8\, d_{15} + d_3\, d_5\, d_{16}$ $-d_3\, d_8\, d_{13} - d_4\, d_5\, d_{15} + d_4\, d_7\, d_{13}$	486
$+\det(F_{11}) = +d_1\,(d_6\, d_{16} - d_8\, d14) - d_2\,(d_5\, d_{16} - d_8\, d_{13}) + d_4\,(d_5\, d_{14} - d_6\, d_{13})$ $= +d_1\, d_6\, d_{16} - d_1\, d_8\, d_{14} - d_2\, d_5\, d_{16}$ $+d_2\, d_8\, d_{13} + d_4\, d_5\, d_{14} - d_4\, d_6\, d_{13}$	486
$-\det(F_{12}) = -[d_1\,(d_6\, d_{15} - d_7\, d_{14}) - d_2\,(d_5\, d_{15} - d_7\, d_{13}) + d_3\,(d_5\, d_{14} - d_6\, d_{13})]$ $= -d_1\, d_6\, d_{15} + d_1\, d_7\, d_{14} + d_2\, d_5\, d_{15}$ $-d_2\, d_7\, d_{13} - d_3\, d_5\, d_{14} + d_3\, d_6\, d_{13}$	486
$+\det(F_{13}) = +d_2\,(d_7\, d_{12} - d_8\, d_{11}) - d_3\,(d_6\, d_{12} - d_8\, d_{10}) + d_4\,(d_6\, d_{11} - d_7\, d_{10})$ $= +d_2\, d_7\, d_{12} - d_2\, d_8\, d_{11} - d_3\, d_6\, d_{12}$ $+d_3\, d_8\, d_{10} + d_4\, d_6\, d_{11} - d_4\, d_7\, d_{10}$	486

$-\det(F_{14}) = -[d_1 (d_7 d_{12} - d_8 d_{11}) - d_3 (d_5 d_{12} - d_8 d_9) + d_4 (d_5 d_{11} - d_7 d_9)]$ $= -d_1 d_7 d_{12} + d_1 d_8 d_{11} + d_3 d_5 d_{12}$ $-d3xd8xd9 - d4xd5xd11 + d4xd7xd9$	486
$+\det(F_{15}) = +d_1 (d_6 d_{12} - d_8 d_{10}) - d_2 (d_5 d_{12} - d_8 d_9) + d_4 (d_5 d_{10} - d_6 d_9)$ $= +d_1 d_6 d_{12} - d_1 d_8 d_{10} - d_2 d_5 d_{12}$ $+d_2 d_8 d_9 + d_4 d_5 d_{10} - d_4 d_6 d_9$	486
$-\det(F_{16}) = -[d_1 (d_6 d_{11} - d_7 d_{10}) - d_2 (d_5 d_{11} - d_7 d_9) + d_3 (d_5 d_{10} - d_6 d_9)]$ $= -d_1 d_6 d_{11} + d_1 d_7 d_{10} + d_2 d_5 d_{11}$ $-d_2 d_7 d_9 - d_3 d_5 d_{10} + d_3 d_6 d_9$	486
The total computational cost of Part 2	3888

On the next step,

$$\text{Adj(E)} = C^T = \begin{bmatrix} +\det(F_1) & -\det(F_5) & +\det(F_9) & -\det(F_{13}) \\ -\det(F_2) & +\det(F_6) & -\det(F_{10}) & +\det(F_{14}) \\ +\det(F_3) & -\det(F_7) & +\det(F_{11}) & -\det(F_{15}) \\ -\det(F_4) & +\det(F_8) & -\det(F_{12}) & +\det(F_{16}) \end{bmatrix} \quad (3.37)$$

Furthermore, let det(E) = g and write E^{-1} from (3.37) as the following:

$$E^{-1} = \begin{bmatrix} +\frac{\det(F_1)}{g} & -\frac{\det(F_5)}{g} & +\frac{\det(F_9)}{g} & -\frac{\det(F_{13})}{g} \\ -\frac{\det(F_2)}{g} & +\frac{\det(F_6)}{g} & -\frac{\det(F_{10})}{g} & +\frac{\det(F_{14})}{g} \\ +\frac{\det(F_3)}{g} & -\frac{det(F_7)}{g} & +\frac{\det(F_{11})}{g} & -\frac{\det(F_{15})}{g} \\ -\frac{\det(F_4)}{g} & +\frac{\det(F_8)}{g} & -\frac{\det(F_{12})}{g} & +\frac{\det(F_{16})}{g} \end{bmatrix} \quad (3.38)$$

For simplifying E^{-1} (3.38), it can be rewritten as below (3.39) and the computational cost of every h_j is 41 CPU cycles. Hence, the computational cost of E^{-1} for 16 elements is 656 CPU cycles.

$$\text{Let } E^{-1} = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 \\ h_5 & h_6 & h_7 & h_8 \\ h_9 & h_{10} & h_{11} & h_{12} \\ h_{13} & h_{14} & h_{15} & h_{16} \end{bmatrix} \quad (3.40)$$

From (3.35), A^T can be written as below (3.41). Next, the resulting matrix of $E^{-1}A^T$ is defined in (3.42). In the following step, the computational cost of every element of $E^{-1}A^T$ is calculated. Elements and their computational costs are listed in Table 3.9.

$$A^T = \begin{bmatrix} c_1 & c_5 & c_9 & c_{13} & c_{17} & c_{21} \\ c_2 & c_6 & c_{10} & c_{14} & c_{18} & c_{22} \\ c_3 & c_7 & c_{11} & c_{15} & c_{19} & c_{23} \\ c_4 & c_8 & c_{12} & c_{16} & c_{20} & c_{24} \end{bmatrix} \tag{3.41}$$

$$E^{-1}A^T = \begin{bmatrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\ e_7 & e_8 & e_9 & e_{10} & e_{11} & e_{12} \\ e_{13} & e_{14} & e_{15} & e_{16} & e_{17} & e_{18} \\ e_{19} & e_{20} & e_{21} & e_{22} & e_{23} & e_{24} \end{bmatrix} \tag{3.42}$$

Table 3.9: The computational cost calculation of $E^{-1}A^T$

Processes	**CCost**
$e_1 = +h_1 c_1 +h_2 c_2 +h_3 c_3 +h_4 c_4$	163
$e_2 = +h_1 c_5 +h_2 c_6 +h_3 c_7 +h_4 c_8$	163
$e_3 = +h_1 c_9 +h_2 c_{10} +h_3 c_{11} +h_4 c_{12}$	163
$e_4 = +h_1 c_{13} +h_2 c_{14} +h_3 c_{15} +h_4 c_{16}$	163
$e_5 = +h_1 c_{17} +h_2 c_{18} +h_3 c_{19} +h_4 c_{20}$	163
$e_6 = +h_1 c_{21} +h_2 c_{22} +h_3 c_{23} +h_4 c_{24}$	163
$e_7 = +h_5 c_1 +h_6 c_2 +h_7 c_3 +h_8 c_4$	163
$e_8 = +h_5 c_5 +h_6 c_6 +h_7 c_7 +h_8 c_8$	163
$e_9 = +h_5 c_9 +h_6 c_{10} +h_7 c_{11} +h_8 c_{12}$	163
$e_{10} = +h_5 c_{13} +h_6 c_{14} +h_7 c_{15} +h_8 c_{16}$	163
$e_{11} = +h_5 c_{17} +h_6 c_{18} +h_7 c_{19} +h_8 c_{20}$	163
$e_{12} = +h_5 c_{21} +h_6 c_{22} +h_7 c_{23} +h_8 c_{24}$	163
$e_{13} = +h_9 c_1 +h_{10} c_2 +h_{11} c_3 +h_{12} c_4$	163
$e_{14} = +h_9 c_5 +h_{10} c_6 +h_{11} c_7 +h_{12} c8$	163

$e_{15} = +h_9\, c_9 + h_{10}\, c_{10} + h_{11}\, c_{11} + h_{12}\, c_{12}$	163
$e_{16} = +h_9\, c_{13} + h_{10}\, c_{14} + h_{11}\, c_{15} + h_{12}\, c_{16}$	163
$e_{17} = +h_9\, c_{17} + h_{10}\, c_{18} + h_{11}\, c_{19} + h_{12}\, c_{20}$	163
$e_{18} = +h_9\, c_{21} + h_{10}\, c_{22} + h_{11}\, c_{23} + h_{12}\, c_{24}$	163
$e_{19} = +h_{13}\, c_1 + h_{14}\, c_2 + h_{15}\, c_3 + h_{16}\, c_4$	163
$e_{20} = +h_{13}\, c_5 + h_{14}\, c_6 + h_{15}\, c_7 + h_{16}\, c_8$	163
$e_{21} = +h_{13}\, c_9 + h_{14}\, c_{10} + h_{15}\, c_{11} + h_{16}\, c_{12}$	163
$e_{22} = +h_{13}\, c_{13} + h_{14}\, c_{14} + h_{15}\, c_{15} + h_{16}\, c_{16}$	163
$e_{23} = +h_{13}\, c_{17} + h_{14}\, c_{18} + h_{15}\, c_{19} + h_{16}\, c_{20}$	163
$e_{24} = +h_{13}\, c_{21} + h_{14}\, c_{22} + h_{15}\, c_{23} + h_{16}\, c_{24}$	163
The total computational cost of $E^{-1}A^T$	3912

Finally, vector r is calculated in (3.43) and the computational cost of each element is calculated and listed in Table 3.10.

$$r = E^{-1}A^T b = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \end{bmatrix} \tag{3.43}$$

Table 3.10: The computational cost calculation of vector r

Processes	**CCost**
$k_1 = +e_1\, b_1 + e_2\, b_2 + e_3\, b_3 + e_4\, b_4 + e_5\, b_5 + e_6\, b_6$	245
$k_2 = +e_7\, b_1 + e_8\, b_2 + e_9\, b_3 + e_{10}\, b_4 + e_{11}\, b_5 + e_{12}\, b_6$	245
$k_3 = +e_{13}\, b_1 + e_{14}\, b_2 + e_{15}\, b_3 + e_{16}\, b_4 + e_{17}\, b_5 + e_{18}\, b_6$	245
$k_4 = e_{19}\, b_1 + e_{20}\, b_2 + e_{21}\, b_3 + e_{22}\, b_4 + e_{23}\, b_5 + e_{24}\, b_6$	245
The total computational cost of vector r	980

When the calculated computational cost of every step of the BTS is summed, the total cost is found as 43852 CPU cycles.

3.3 Experiment Results of The BTS

In the original paper [46], not only the BTS system is discussed, but also the details of the experiments that had been done is also explained. Although the simple geometric model and the straightforward mathematical calculations of the BTS has built an expectation as it would achieve perfect accuracy in the location estimation process, test results revealed as it has gotten close to the actual location only 34.5 meters (Root mean Squire Error). Most probably this deflection is reasoned by the following assumptions that had been done while the technical necessities of the solution system are decided:

- The SAGE algorithm should measure aod and aoa angles perfectly through antenna arrays used on both BS and MO [38].
- The two-step proximity detection algorithm must allow the system to distinguish the LOS and SB-NLOS arriving signals correctly and ignore the rest [40].

Chapter 4

PROPOSED MODEL OF SOLUTION

4.1 Technical Properties of The Model

Our solution requires the cost-effectiveness and speed of LOS systems and the accuracy of NLOS systems as they have been reviewed in the literature. This model has BSs with known positions and SCs and an MO with unknown positions. The clocks of the BSs are assumed to be synchronised with each other but not with the clock of MO; by using the TDOA technique, the total distance travelled by each signal through a BS, to an SC, and then to the MO can be calculated. It is also assumed that the MO and the BSs have antenna arrays, so, by using SAGE-like [38] algorithms, the AOD, AOA, and TDOA values can be measured [39]. These known and calculated elements were used to construct our geometric model of the solution. Signals in real-life broadcasting have a multipath nature, so they can reach their destination in many ways, including directly, scattered with single-bounce, or scattered with many bounces. On every bounce scattering, signals lose their power and start to delay. In this study, only the direct and single bounce scattered arriving signals were considered. As described in Seow and Tan [40], a two-step proximity detection algorithm was used to distinguish and ignore the rest.

4.2 Geometric Model

Our model contains five components illustrated in Figure 4.1.

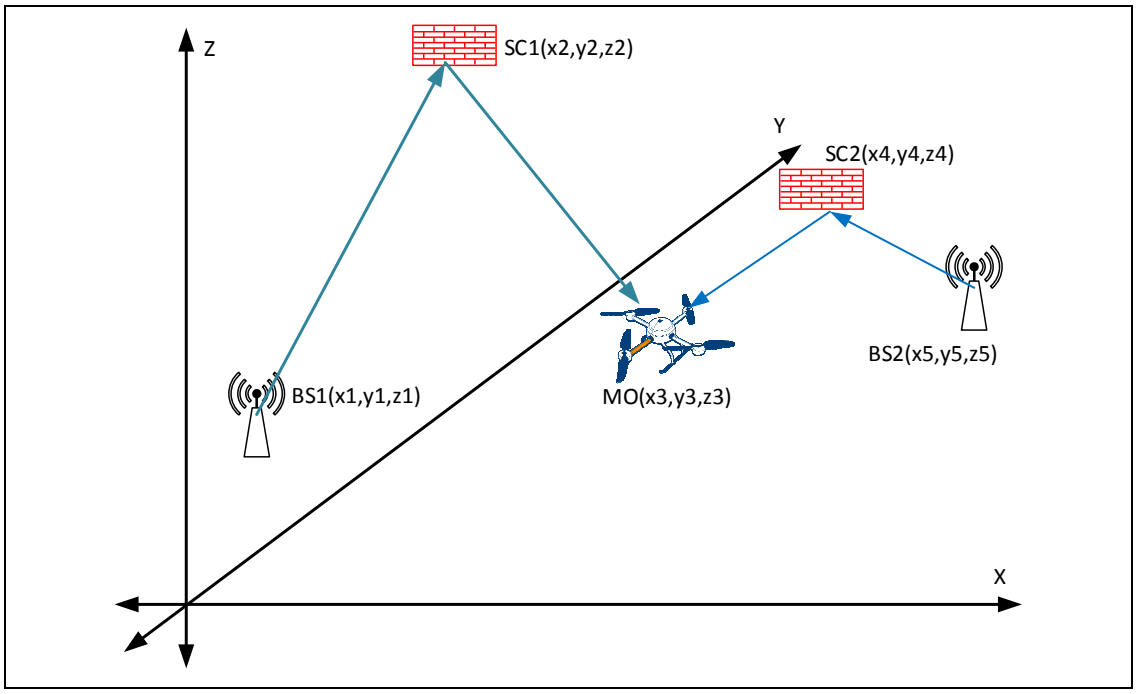

Figure 4.1: Components of the graphical model

These components are the MO, two BSs, and two SCs. To simplify this model, the techniques described in Fang [41] were used: A set of local, right-handed orthogonal axes was chosen and the nearest BS was carried to the origin to become BS_1 (0, 0, 0). One of the axes was used as the station baseline (i.e., the x-axis), and the other axes were placed orthogonal to the x-axis. Then, the whole system and its components were shifted and rotated until the second BS was placed on the x-axis. The remaining components' coordinates in the simplified model are therefore $BS_2(xb_2, 0, 0)$, $SC_1(xs_1, ys_1, zs_1)$, $SC2(xs_2, ys_2, zs_2)$, and $MO(x_m, y_m, z_m)$. Figure 4.2 illustrates the model after this geometric transformation, with transformed points and rotated axes.

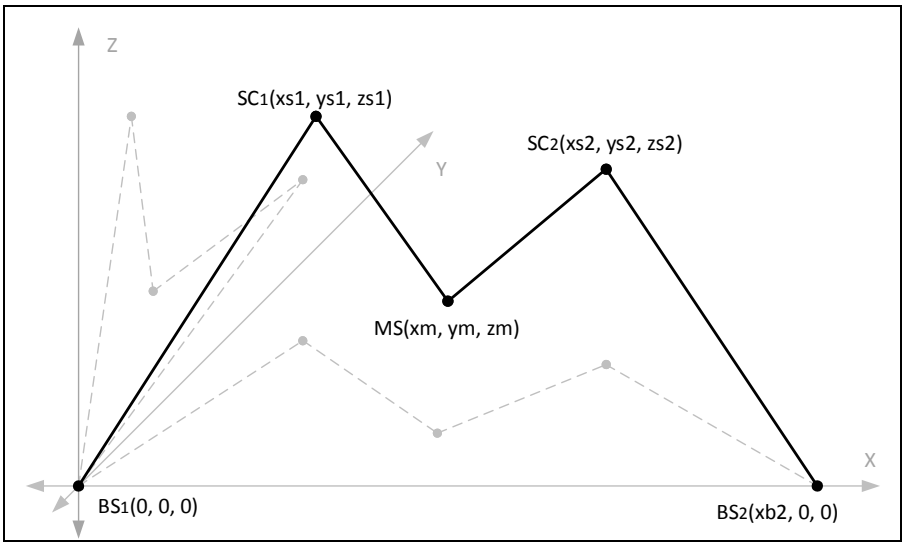

Figure 4.2: Geometrically Transformed Model

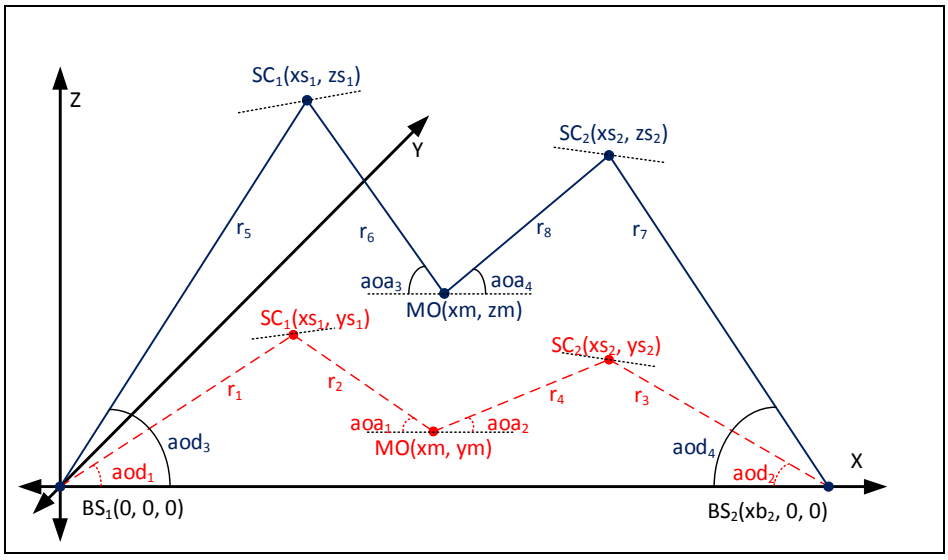

Figure 4.3: Elements of the geometrically transformed model

In Figure 4.3, the elements of the geometrically transformed model are presented on X-Y and X-Z planes. The angles of arrival (aoa_1, aoa_2, aoa_3, aoa_4), angles of departure (aod_1, aod_2, aod_3, aod_4), and the coordinates of BS_1 and BS_2 are known (see Section 4.1). The total travel time of each signal through the BSs, to the SCs, and then to the MO can be calculated using TDOA (see Section 2.2 and Section 2.3); therefore, the total distance travelled by the signals can also be calculated (DL_1, DL_2). However, the locations of the SCs and the mobile object MO are unknown.

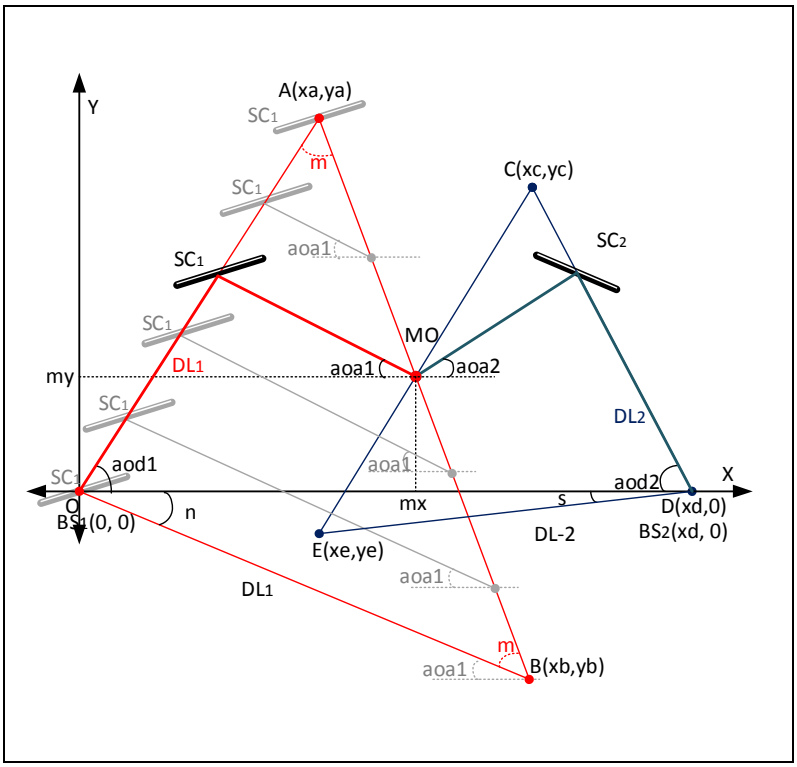

Figure 4.4: Location estimation of MO with LOS and SBS-NLOS in 2D space

Figure 4.4 illustrates the relations between the elements of the model on the X–Z and X–Y planes. DL_1 is the calculated distance travelled by the signal transmitted from BS1 to the MO. This signal can reach the MO directly, without bouncing any scatter; this is the LOS part of the model. There is also the SBS NLOS part where the signal bounces with a scatter on its route. On the X–Y plane, r_1 and r_2 are the distances from BS_1 to SC_1 and from SC_1 to MO, respectively; their sum must be equal to DL_1. The sum of r_5 and r_6 on the X–Z plane must also be equal to DL1. Although the magnitudes of r_1, r_2, r_3, r_4, r_5, r_6, r_7, and r_8 are unknown, the equations can be written with the calculated magnitudes of DL_1, and DL_2 as follows:

$$DL_1 = r_1 + r_2 = r_5 + r_6. \qquad DL_2 = r_3 + r_4 = r_7 + r_8 \tag{4.1}$$

To develop an applicable solution, the projections of the components onto the X–Y plane are focused on first. As it is illustrated in Figure 4.4, this approach delivers a

solution in 2D space. Then, projecting the components onto the X–Z plane brings the solution to the 3D space (See Figure 4.5).

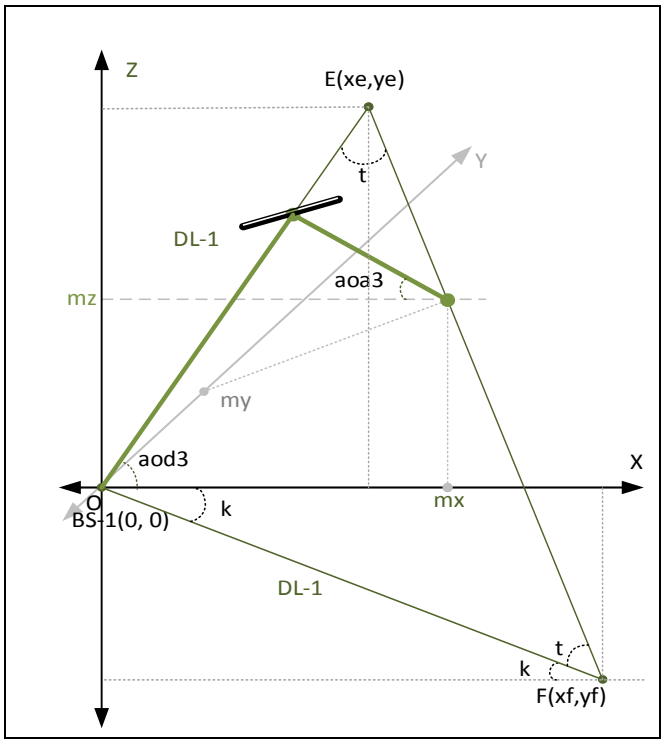

Figure 4.5: Location estimation on X-Z plane

As demonstrated in Figure 4.4, the SC_1 can be located anywhere on the line |OA| and the MO must be placed on the line |AB|. The first LOS case is when SC_1 and the MO share a location at point A. Since DL_1 is the sum of r_1 and r_2, r_1 should be equal to DL_1 and r_2 should be equal to zero. The second LOS case places the MO on point B, and BS_1 and SC_1 share a location on the origin (O) of |OA|. In this case, r_1 must be zero and r_2 must be equal to DL_1. Other positions on |AB| represent the SBS NLOS cases since they require $r_1 > 0$ and $r_2 > 0$. In Figure 4.4, two BS-SC-MO sets are merged to estimate the MO position on the X–Y plane. The location of the MO must be at the intersection point of the lines |AB| and |CE|. The angles n, s, and k in Figure 4.4 and Figure 4.5 are equal to the angles aoa1, aoa2, and aoa3, respectively, so the coordinates of A, B, C, D, and E can be calculated and the equations for |AB|, |DC|, and |EF| can

be written. MO (xm, zm) must be a point on the line |EF|, and since xm is already estimated, the last unknown of the model, zm, can be calculated using the equations for |EF| and xm.

Chapter 5

ALGORITHMS FOR ESTIMATION

5.1 Introduction

Based on the literature review conducted, this study aimed to develop a solution for the location estimation problem that is software and hardware-oriented, computationally cost-effective, fast, and accurate. Consequently, we decided to use CORDIC methods within the solution algorithms. Three well-known CORDIC methods serve this purpose: the original CORDIC and two faster variations of it, known as the fixed angle rotation method (FARM) [42, 43] and the dynamic angle rotation method (DARM) [44, 47]. They are cost-effective substitutes for trigonometric functions. In this part of the work, a set of algorithms we named ACE (Arif-Cem-Emre) were developed and constructed on the geometrically transformed model, and its components are described in Section 4.2.

5.2 ACE Algorithms

The ACE algorithm is composed of eight main steps. The first five are for finding the coordinates of the MO on the X–Y plane in 2D space. The remaining three steps are for extending the 2D solution to the 3D space by including the X–Z pane and its components. In order to speed up and reduce the computational costs of calculations, all heavy weighted trigonometric functions are decided to be replaced with CORDIC algorithm. Salamah and Doukhnitch [42] showed that the original CORDIC algorithm has some performance leakages, as it requires numerous sub-iterations and a factor compensation process on every main iteration. FARM and DARM use wiser

algorithms that exclude unnecessary iterations and compensation processes from each rotation. Both FARM and DARM use recursive iterations instead of their ancestor. For more detailed information See Section 2.4.3 for FARM, and Section 2.4.4 for DARM.

Three methods are used in ACE: To emulate the trigonometric functions FARM [44] and DARM [47] are used. Furthermore, a completely new vector-breaking method (VBM) were developed and used in this study by the vector lengthening method [44].

The ACE algorithm is implemented with FARM and DARM separately. As a result, two versions of the ACE are developed: ACE-FARM and ACE-DARM.

5.2.1 ACE-FARM Algorithm

As it is extensively explained in Section 2.4.3, FARM is using a fixed stepping angle in repetitive rotation while converging to its target. All necessary trigonometric functions are replaced with FARM methods in the ACE algorithm. The implementation steps of the ACE-FARM method are depicted in Figure 5.1. FARM's step sizes are determined by a fixed value (angle in radians) $\sigma = \arcsin(2^{-k})$, where k is a parameter to decide the level of accuracy. M is the circular rotation matrix [47]. Here, $u = 2^{-(2k+1)}$, $v = 2^{-k}$, and $s = \pm 1$ is the operators used to decide the direction of rotation. Using basic arithmetic operations and binary shifts instead of trigonometric functions in any rotation or coordinate calculation reduces the computational cost of the process. Therefore, FARM is more useful in applications where there are limited hardware resources

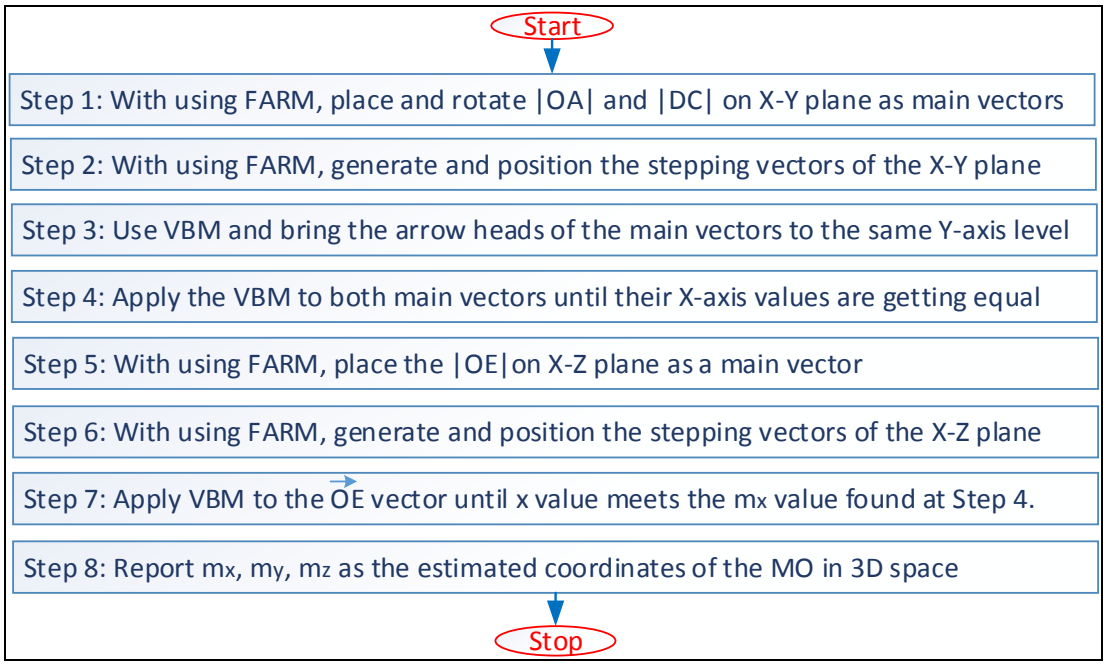

Figure 5.1: General scheme of ACE-FARM algorithm

$$M(j) = \begin{pmatrix} \cos\sigma & \sin\sigma \\ -\sin\sigma & \cos\sigma \end{pmatrix} = \begin{pmatrix} (1-2^{-(2k+1)} & 2^{-k} \\ -2^{-k} & (1-2^{-(2k+1)}) \end{pmatrix} \tag{5.1}$$

The vector coordinates are changing recursively as follows:

$$V(j+1) = \begin{pmatrix} x_{j+1} = x_j - x_j u + s y_j v \\ y_{j+1} = y_j - y_j u - s x_j v \end{pmatrix} \tag{5.2}$$

The FARM is used for placing the arrowhead of the related vector to the right coordinates for emulating the trigonometric functions. The method is using the definitions of (5.1) and (5.2) and it is following the instructions of the algorithm shown in Fig. 5.1.

Here the variables $\varepsilon = 2^{-k}$ is the acceptable level of error, k is the parameter for adjusting the accuracy level, α is the targetted angle in radian which the vector has to be rotated, β_i is the current rotated total angle, and σ is the fixed step rotation angle. The dl is the length of the vector.

Algorithm 5.1: Emulating the sin, cos functions in ACE-FARM

Input: α, σ, dl

Output x_j, y_j

Begin

$\beta_i = 0$

While $((\alpha - \beta_i) > \varepsilon)$

Rotate d_1 by stepping angle σ

End While

End

In the first step of the ACE-FARM, the vector $\overrightarrow{OA}$ is placed and then it is rotated with angle aod_1 on the X-Y plane using Algorithm 5.1. This process is similar to the vector rotation as depicted in Figure 2.9. Here, the vector dl is DL_1, and α is aod_1. The DL_1 is placed on x-axis starting from origin (O), and then, it is gradually rotated on the counterclockwise direction with a fixed stepping angle σ until the total rotation angle is reached to aod1 which is the targeted angle (Rotation loop stops when $(aod_1 - \beta_i) < \varepsilon$).

5.2.2 ACE-DARM Algorithm

For decreasing the number of iterations while achieving the desired accuracy level, DARM changes the rotation angle dynamically in recursive iterations. You can have extensive details of the DARM in Section 2.4.4. The steps of the ACE-FARM as presented in Figure 5.1 are implemented in ACE-DARM with modified step 1, step 2, step 5, and step 6. In these steps, FARM methods are replaced with DARM methods. Hence, vector lengthening, and rotation processes are done with DARM methods.

ACE-DARM uses non-fixed rotation angle $\sigma i = \arcsin(2^{-k})$ as k=5..11 and i=1..7. Figure 2.10 presents how DARM is approaching the targeted angle σ_T. Recursive rotations starts with $\sigma_5 = \sigma(5) = \arcsin(2^{-5})$ and continues until it exceeds σ_T or $|\sigma_T - \sigma_{t(j)}| < \epsilon$ where $\epsilon < 2^{-5}$ (error) and $\sigma_{t(j)}$ is the sum of rotated angles. In the case, the target

angle is exceeded, the stepping angle $\sigma_{k+1} = \arcsin(2^{-k+1})$ is recalculated with advancing the accuracy parameter k to k+1 and recursive angle rotations restart from the previous step as $\sigma t_{(j)} < \sigma_T$. These recursive loops continue until $\sigma t_{(j)}$ approaches enough to the target angle where the condition $|\sigma_T - \sigma t_{(j)}| < \epsilon$ is satisfied.

5.2.3 Vector Breaking Method (VBM)

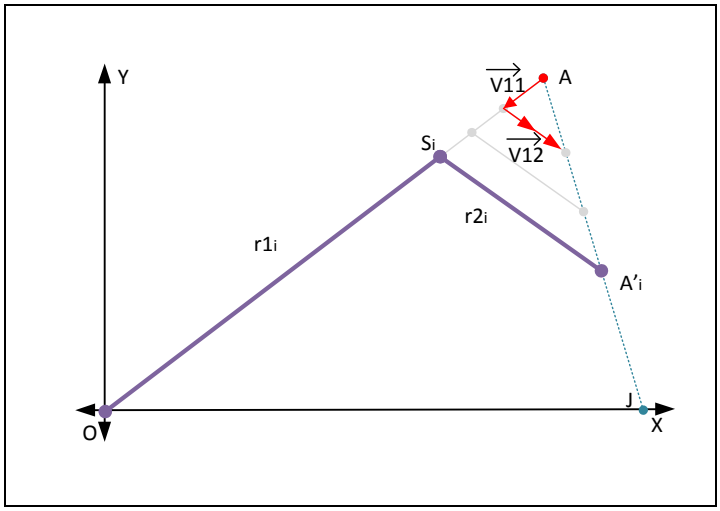

Figure 5.2: Vector Breaking Method

As discussed in Section 4.2 and illustrated in Figure 4.4, points A and C represent the possible first cases of LOS MO positions. As shown in Figure 5.2, if the MO is placed on point A, from (4.1) it is required that $r_1 = DL_1$, and $r_2 = 0$. When the MO is placed in SBS NLOS case, it requires $r_1 < DL_1$, $r_2 > 0$.

When they are drawn on the same graph, it looks as though the vector |OA| has been broken into two pieces, r_1, and r_2, and hence the name 'vector-breaking method' (VBM). The method shortens r_1 and lengthens r_2 while keeping the magnitude, DL_1, constant. For this purpose, the size of r_1 is reduced by subtracting the stepper vector V_{11} from r_1 on every turn of the iteration. To keep DL_1 constant, the second stepper vector V_{12} is recursively added to r_2 within an inner loop. The stepper vectors V_{11} and

V_{12} are horizontally μ meters lengthened vectors which are then rotated with angles aod_1 and aoa_1, respectively. μ is the second parameter, along with k, that has a significant effect on the accuracy of the model. The VBM is presented in Figure 5.2 and the implementation is described in Algorithm 5.2.

Algorithm 5.2: Vector breaking method (VBM)

Input: T(a targeted point on $\overrightarrow{AX}$ vector), $\overrightarrow{V_{11}}, \overrightarrow{V_{12}}$ (Stepping vectors)
Output x_j, y_j // coordinates of A'_i

Begin

$A'_0 = A$
$r_{1_0} = DL_1$
$r_{2_0} = 0$
i = 0
j = 0
While ((T - A'_i) > ε)
    $r_{1_i} = r_{1_i} - \overrightarrow{V_{11}}$
    i = i + 1
    While (DL_1-($r_{1_i} + r_{2_j}$)) > ϵ)
        $r_{2_j} = r_{2_j} + \overrightarrow{V_{12}}$
        j = j + 1
    End While
End While

End

Chapter 6

SIMULATIONS

The graphical model described in Section 4.2 and the algorithms described in Section 5.2 were tested for accuracy and computational cost. For this purpose, three applications were developed and executed in MATLAB (MathWorks, Inc., Natick, MA, USA) and their codes are at the Appendix part of the thesis.

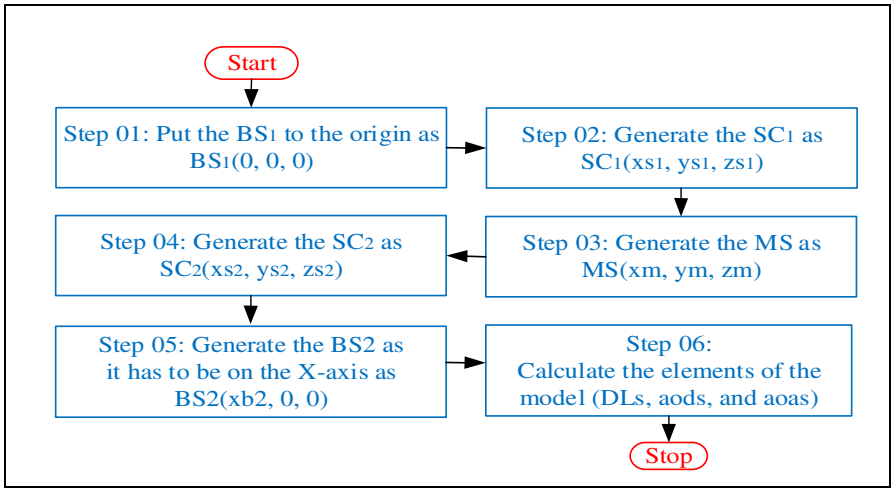

Figure 6.1: Sample Generator Algorithm

The first application follows an algorithm (see Figure 6.1) to generate sample sets of the graphical model components (BS_1, BS_2, SC_1, SC_2, and MO) and elements (DL_1, DL_2, aod_1, aod_2, aod_3, aod_4, aoa_1, aoa_2, aoa_3, and aoa_4) that are illustrated in Figure 4.3. They are randomly placed into a 3D space bounded at 500 m on the x-axis, 500 m on the y-axis, and 100 m on the z-axis. The second application was developed using the ACE-FARM explained in Section 5.2.1 and its steps of implementation are illustrated

in Figure 5.1. For implementing the ACE-DARM, the third application was developed as described extensively in Section 5.2.2.

Both ACE algorithms were tested at 95% confidence levels. The tests were repeated with different accuracy levels (k) for ACE-FARM to comprehend the influence of chosen values of k on the resulting errors and computational costs of the processes. As mentioned in Section 2.4.3 and Section 2.4.4, k is a significant parameter in vector rotation and lengthening, and therefore has an important role in both ACE variations.

The weights of the operations, shown in Table 3.1, were used for calculating the computational cost of each process [45]. For each sample in the set, the calculated computational cost and the errors in the distance between the estimated and actual coordinates of the MO were recorded to evaluate the accuracy and the computational efficiency of both ACE algorithms.

6.1 Simulation Results

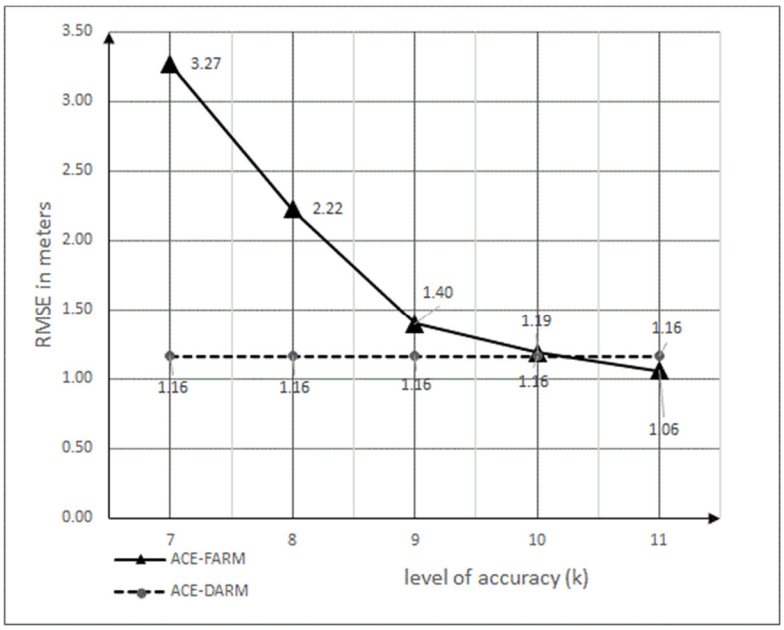

Figure 6.2: Root Mean Square Error (RMSE) for ACE-FARM and ACE-DARM

The recorded accuracies and computational costs of ACE-FARM and ACE-DARM with different k values are illustrated in Figure 6.2 and Figure 6.3. Figure 6.2 presents the exponentially improving accuracy of the ACE-FARM algorithm as k is incrementally increased. Since, k is automatically incremented in ACE-DARM, while rotation stepping angle is dynamically changing in iteration, it gives a fixed accuracy value (Root mean squire error).

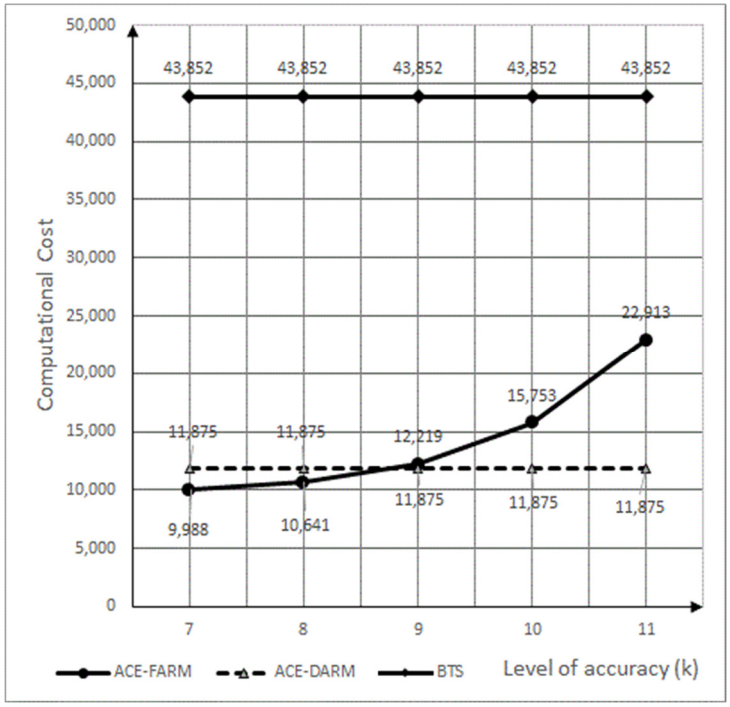

Figure 6.3: Computational Costs of ACE-FARM, ACE-DARM, and BTS.

Figure 6.3 shows the expected side effect of increasing k, that is, the exponentially rising computational costs in ACE-FARM.

As explained in Section 2.1, the FCC's 2001 E911 accuracy requirements have been accepted as mandatory standards in many studies on location estimation. However, the 2019 declaration of the FCC [52] requires an important revision to the existing localisation techniques to fulfil the new standards. Many techniques which satisfied

the FCC requirements before the update have similar working conditions to ACE in that they (a) use an SBS NLOS signal arriving approach and (b) perform location estimation on the MO and examine the accuracy of their estimations. The root mean square error of the solution by (BTS) [46] is 34.5 meters. The solution by Khajehnouri and Sayed (KSS) [22] has a mean error of 25 meters at its best performance. The direction of arrival-based localisation algorithm developed by Li and Lu [53] presented estimation errors (400 m × 400 m horizontal scale) of 16 m under its best conditions. The weighted centralized cooperative single-bounced scattering NLOS technique by Liu, Zhu, Jiang, and Huang [54] has an average location estimation error of 12 m.

In recent studies [55, 56], more accurate localisation techniques have been introduced; however, they all are designed to work on the BSs and therefore are not comparable to our study. As the test results are shown in Figure 6.2 reveal, ACE-DARM and ACE-FARM satisfied the new FCC rules under all conditions of k.

One of the best-known features of the CORDIC algorithm is the computational efficiency it provides while imitating trigonometric functions. To prove our solution is a nontrivial contribution to the field, we decided that the BTS would be used as the benchmark by which to measure our algorithm since it uses matrix calculations to estimate the position of the MO and is expected to present better computational efficiency than the other techniques referred to above. Since the original study does not include metrics on its computational cost, we considered the weights of the operations given in Table 3.1 along with the methods and calculations described in its text and calculated its computational cost (see Section 3.1). We found the cost of BTS to be 43,852, which is more than four times the cost of ACE-FARM under its best condition of k = 7 and almost four times the cost of ACE-DARM. The computational

cost comparison between ACE variations and BTS is demonstrated in Figure 6.3. The presented efficiency and precision of the ACE algorithm make it a significant option among its alternatives.

Chapter 7

CONCLUSION

In this thesis, the details of two novel location estimation techniques are given. The first contribution is a new algorithm, named ACE, which (1) allows a MO to self-position using the fixed angle rotation method (FARM) and dynamic angle rotation method (DARM) of the CORDIC algorithm with both LOS and SBS NLOS signal arriving approaches and (2) is a hardware-oriented solution, as the CORDIC method performs all calculations with basic binary shift and add operations supported by all CPUs since 1956. The second contribution of the study is a new calculation method, VBM, used for estimating the location of an MO using the signals bounced from scatters with unknown locations. CORDIC is used for the first time with SBS NLOS signal arriving approach, and this is the third contribution of the study.

Both ACE variations performed remarkably well relative to the accuracy and computational costs consented to in the field of research. The FARM and DARM variations of the CORDIC algorithm were used in our study to speed up and reduce the cost of computations.

In future studies, ACE variations can be extended to work with multi-bounced scattering NLOS and LOS arriving signals. Even though in simulations, ACE-FARM and ACE-DARM presented outstanding performances, these results are not enough for

making the final decision. They must also be tested in the field with real-life conditions.

REFERENCES

[1] H. Choi, M. Geeves, B Alsalam B, F. Gonzalez, "Open source computer-vision based guidance system for UAVs on-board decision making," 2016 IEEE Aerospace Conference; Big Sky, MT, USA; 2016. pp. 1-5. DOI: 10.1109/AERO.2016.7500600

[2] J.L. Rullan-Lara, S. Salazar, R. Lozano, UAV real-time location using a wireless sensor network. In: 2011 8th Workshop on Positioning, Navigation and Communication; Dresden, Germany; 2011. pp. 18-23. DOI: 10.1109/WPNC.2011.5961008

[3] H. Tsuji, D. Gray, M. Suzuki, R. Miura, Radio location estimation experiment using array antennas for high altitude platforms. In: 2007 IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications, Athens, Greece; 2007. pp. 1-5. DOI: 10.1109/PIMRC.2007.4394608

[4] R. Abbas, K. Michael, COVID-19 Contact Trace App Deployments: Learnings From Australia and Singapore. IEEE Consumer Electronics Magazine 2020; 9 (5): 65-70. DOI: 10.1109/MCE.2020.3002490

[5] Y. Weizman, A.M. Tan, F.K. Fuss, Use of wearable technology to enhance response to the Coronavirus (COVID-19) pandemic. Public Health 2020; 185: 221-222. DOI: 10.1016/j.puhe.2020.06.048.

[6] S.T.S. Thong, C.T. Han, T.A. Rahman, Intelligent fleet management system with concurrent GPS & GSM real-time positioning technology. In: 2007 7th International Conference on ITS Telecommunications; Sophia Antipolis, France; 2007. pp. 1-6, DOI: 10.1109/ITST.2007.4295849.

[7] R. Zantout, M. Jrab, L. Hamandi, and F. Sibai FN, Fleet management automation using the global positioning system. In: 2009 International Conference on Innovations in Information Technology (IIT); Al Ain, United Arab Emirates; 2009. pp. 30-34. DOI: 10.1109/IIT.2009.5413792

[8] J. Du, M.J. Barth, Next-Generation Automated Vehicle Location Systems: Positioning at the Lane Level, in IEEE Transactions on Intelligent Transportation Systems, vol. 9, no. 1, March 2008.pp. 48-57. DOI: 10.1109/TITS.2007.908141

[9] Federal Communications Commission. Fact sheet: FCC wireless 911 requirements. Government Printing Office (accessed 5 May 2001). Website http://www.fcc.gov/e911/

[10] E.D. Seeman, M.T O'Hara, J. Holloway, A. Forst, The impact of government intervention on technology adoption and diffusion: the example of wireless location technology. Electronic Government, an International Journal, 2007; 4 (1): 1-19.

[11] D. Mohapatra, S.B. Suma, Survey of location-based wireless services. IEEE International Conference on Personal Wireless Communications; ICPWC, New Delhi, India; 2005. 358-362.

[12] D. Dao, C. Rizos, J. Wang, Location-based services: technical and business issues. Gps Solutions, 2002; 6 (3): 169-178.

[13] H.M Yunos, J.Z. Gao, S. Shim, Wireless advertising's challenges and opportunities. Computer, 2003; 36 (5): 30-37.

[14] L.A. Stilp, Carrier and end-user applications for wireless location systems. Wireless Technologies and Services for Cellular and Personal Communication Services (International Society for Optics and Photonics), January 1996; Vol. 2602: 119-126.

[15] R. Want, B. Schilit, Expanding the horizons of location-aware computing. Computer, 2001; 34 (8): 31-34.

[16] A. Tsalgatidou, J. Veijalainen, J. Markkula, A. Katasonov, S. Hadjiefthymiades, Mobile e-commerce and location-based services: Technology and requirements. ScanGIS, June 2003; Vol. 2003: 1-14.

[17] S. Gezici, A survey on wireless position estimation. Wireless personal communications, 2008; 44 (3): 263-282. [18] Sayed AH, Tarighat A, Khajehnouri N. Network-based wireless location: challenges faced in

developing techniques for accurate wireless location information. IEEE signal processing magazine, 2005; 22 (4): 24-40.

[19] J.J. Caffery, A new approach to the geometry of TOA location. Vehicular Technology Conference Fall 2000. IEEE VTS Fall VTC2000. 52nd Vehicular Technology Conference; Boston, MA, USA; 2000. (Cat. No. 00CH37152); Vol. 4: 1943-1949.

[20] J.H. Reed, K.J. Krizman, B.D. Woerner, T.S. Rappaport, An overview of the challenges and progress in meeting the E-911 requirement for location service. IEEE Communications Magazine, 1998; 36 (4): 30-37.

[21] M.A. Spirito, A.G. Mattioli, Preliminary experimental results of a GSM mobile phones positioning system based on timing advance. Gateway to 21st Century Communications Village. VTC 1999-Fall. IEEE VTS 50th Vehicular Technology Conference, Amsterdam, Netherlands; September 1999. (Cat. No. 99CH36324); Vol. 4: 2072-2076.

[22] N. Khajehnouri, A.H. Sayed, A non-line-of-sight equalization scheme for wireless cellular location. In 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, Hong Kong, China; April 2003. Proceedings:(ICASSP'03); Vol. 6: VI-549.

[23] I. Guvenc, C.C. Chong, A survey on TOA based wireless localization and NLOS mitigation techniques. IEEE Communications Surveys & Tutorials, 2009; 11 (3): 107-124.

[24] S. Al-Jazzar, J. Caffery, H.R. You, A scattering model-based approach to NLOS mitigation in TOA location systems. IEEE 55th Vehicular Technology Conference. VTC, Birmingham, AL, USA; Spring 2002. (Cat. No. 02CH37367); Vol. 2: 861-865.

[25] Z. Wang, S.A. Zekavat, Omnidirectional mobile NLOS identification and localization via multiple cooperative nodes. IEEE Transactions on Mobile Computing, 2011; 11 (12): 2047-2059.

[26] I.J., Paton, E.W. Crompton, J.G. Gardiner, J.M. Noras, Terminal self-location in mobile radio systems. Sixth International Conference on Mobile Radio and Personal Communications, Coventry, UK; December 1991: 203-207.

[27] Y.T. Chan, K.C. Ho, A simple and efficient estimator for hyperbolic location. IEEE Transactions on signal processing, 1994; 42 (8): 1905-1915.

[28] C. Botteron, A. Host-Madsen, M. Fattouche, Cramer-Rao bounds for the estimation of multipath parameters and mobiles' positions in asynchronous DS-CDMA systems. IEEE Transactions on Signal Processing, 2004; 52 (4): 862-875.

[29] A. Kangas, I. Siomina, T. Wigren, Positioning in LTE. Handbook of Position Location: Theory, Practice, and Advances: John Wiley and Sons, Inc., Publication, 2011: 1081-1127. DOI:10.1002

[30] Y. Zhao, Standardization of mobile phone positioning for 3G systems. IEEE Communications Magazine, 2002; 40 (7): 108-116.

[31] K. W. Cheung, H.C. So, W.K. Ma, Y.T. Chan, Least squares algorithms for time-of-arrival-based mobile location. IEEE Transactions on Signal Processing, 2004; 52 (4): 1121-1130.

[32] N. Patwari, J. N. Ash, S. Kyperountas, A. O. Hero, R. L. Moses, et al., Locating the nodes: cooperative localization in wireless sensor networks. IEEE Signal processing magazine, 2005; 22 (4): 54-69.

[33] T. Roos, P. Myllymäki, H. Tirri, P. Misikangas, J. Sievänen, A probabilistic approach to WLAN user location estimation. International Journal of Wireless Information Networks, 2002; 9 (3): 155-164.

[34] T. Roos, P. Myllymaki, H. Tirri, A statistical modelling approach to location estimation. IEEE Transactions on Mobile Computing, 2002; 1 (1): 59-69.

[35] C. Mazenc, X. Merrheim, J.M. Muller, Computing Functions cos/sup-1/and sin/sup-1/Using CORDIC. IEEE Transactions on Computers, 1993; 42 (1): 118-122.

[36] T. BingJuang, H. FengLin, CORDIC algorithm for vectoring mode without constant scaling factors. Electronics Letters, 1999; 35 (12): 971-972.

[37] J. Villalba, T. Lang, E.L. Zapata, Parallel compensation of scale factor for the CORDIC algorithm. Journal of VLSI signal processing systems for signal, image and video technology, 1998; 19 (3): 227-241.

[38] J.A. Fessler, A.O. Hero, Space-alternating generalized expectation-maximization algorithm. IEEE Transactions on signal processing, 1994; 42 (10): 2664-2677.

[39] S. Sakagami, S. Aoyama, K. Kuboi, S. Shirota, A. Akeyama, Vehicle position estimates by multi-beam antennas in multipath environments. IEEE Transactions on Vehicular Technology, 1992; 41 (1): 63-68.

[40] C.K Seow, S.Y. Tan, Non-line-of-sight localization in multipath environments. IEEE Transactions on Mobile Computing, 2008; 7 (5): 647-660.

[41] B.T. Fang, Simple solutions for hyperbolic and related position fixes. IEEE transactions on aerospace and electronic systems, 1990; 26 (5): 748-753.

[42] M. Salamah, E. Doukhnitch, An efficient algorithm for mobile objects localization. International Journal of Communication Systems, 2008; 21 (3): 301-310.

[43] E. Doukhnitch, M. Salamah, A. Sandouka, Novel hardware-oriented algorithms for TDOA positioning technique in cellular networks. Mathematical Methods in Engineering, Springer, Dordrecht, Netherlands; 2007: 347-357.

[44] E. Doukhnitch, M. Salamah, E. Ozen, An efficient approach for trilateration in 3D positioning. Computer communications, 2008; 31 (17): 4124-4129.

[45] J.M. Muller, Elementary functions Algorithms and Implementations, Springer Science+Business Media New York, USA; 2016. DOI: 10.1007

[46] B.Y. Shikur, T. Weber, Tdoa/aod/aoa localization in nlos environments. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Florence, Italy; (2014, May): 6518-6522.

[47] E. Doukhnitch, M. Salamah, General approach to simple algorithms for 2-D positioning techniques in cellular networks, Computer Communications, Volume 31, Issue 10, 2008, ISSN 0140-3664, Pages 2185-2194.

[48] I. Jami, M. Ali, R.F. Ormondroyd, Comparison of Methods of Locating and Tracking Cellular Mobiles, Novel Methods of Location and Tracking of Cellular Mobiles and Their System Applications (Ref. No. 1999/046), IEE Colloquium, London UK, 1/1-1/6.

[49] W. Xinning, N. Palleit, T. Weber, ”AOD/AOA/TOA-based 3D positioning in NLOS multipath environments,” 2011 IEEE 22nd International Symposium on Personal, Indoor and Mobile Radio Communications, Toronto, ON, Canada; 2011.pp. 1289-1293. DOI: 10.1109/PIMRC.2011.6139709

[50] S. Samadi, M.R. Khosravi, J.A. Alzubi, O.A. Alzubi, V.G. Menon, ”Optimum range of angle tracking radars: theoretical computing.” International Journal of

Electrical and Computer Engineering (IJECE) Vol 9, No 3, June 2019, DOI: 10.11591/ijece.913, pp. 1765-1772

[51] A. Roxin, J. Gaber, M. Wack, Nait-Sidi-Moh . ”Survey of wireless geolocation techniques.” 2007 IEEE Globecom Workshops. IEEE, Washington, DC, USA; 2007.

[52] Federal Communications Commission. Fact sheet: FCC wireless 911 Location Accuracy requirements. Government Printing Office (accessed 16 January 2020). Website https://www.federalregister.gov/documents/2020/01/16/2019-28483/wireless-e911-location-accuracyrequirements.

[53] J. Li, I. Lu, J.S. Lu, DOA-based localization algorithms under NLOS conditions. IEEE Long Island Systems, Applications and Technology Conference (LISAT). IEEE, Farmingdale, NY, USA; 2018 May. pp. 1-6.

[54] B. Liu, X. Zhu, Y. Jiang, Y. Huang, Low Complexity Cooperative Positioning in Multipath Environment. In 2018 IEEE/CIC International Conference on Communications in China (ICCC). IEEE, Beijing, China; 2018 August. pp. 298-303.

[55] H. Liu, W. Dai, Y. Shen, Super-resolved Localisation without Identifying LoS/NLoS Paths. arXiv preprint. 2019 November, arXiv:1910.12662.

[56] C. Xu, Z. Wang, Y. Wang, Z. Wang, L. Yu, Three passive TDOA-AOA receivers-based flying-UAV positioning in extreme environments. IEEE Sensors Journal, 2020; 2 (16): 9589-9595.

APPENDICES

Appendix A: MATLAB Source Code of Samples Set Generator

```
clc;
clear;
%---- Parameters for Sampling ------------------------
coef=6.0000;        %Coefficient value
ss=1000;            % Sample Size
a_max=85; a_min=5; % Angle ranges

%----- File names ----------------------------------
s_date='20191226';
fn_t = strcat(pwd,'\','SampleSet-',s_date,'-SS',int2str(ss),'-t.txt');
fn_d = strcat(pwd,'\','SampleSet-',s_date,'-SS',int2str(ss),'-d.txt');
%-----------------------------------------------------
%fw=fopen(strcat(pwd,'\','samples_2_BSs_2019b_t.txt'),'w');
%fw2=fopen(strcat(pwd,'\','samples_2_BSs_2019b_d.txt'),'w');
fw=fopen(fn_t,'w');
fw2=fopen(fn_d,'w');

%---- Print Table Header -----------------------------
fprintf(fw,'%8s\t%8s\t%8s\t%8s\t%8s\t',' x_B1',' y_B1',' z_B1',' r1','
aod1');
fprintf(fw,'%8s\t%8s\t%8s\t%s\t%s\t%s\t',' x_S1',' y_S1',' z_S1',' r2',' aoa1','
dl1');
fprintf(fw,'%8s\t%8s\t%s\t%s\t%s\t',' x_M',' y_M',' z_M',' r3',' aoa2');
fprintf(fw,'%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t',' x_S2',' y_S2',' z_S2',' r4','
aod2',' dl2');
fprintf(fw,'%8s\t%8s\t',' x_B2',' y_B2',' z_B2');
fprintf(fw,'%8s\t%8s\t%8s\t%8s\t%8s\t',' r5',' r6',' dl3',' aod3',' aoa3');
fprintf(fw,'%8s\t%8s\t%8s\t%8s\t%8s\t\n',' r7',' r8',' dl4',' aod4',' aoa4');


%---- Loop for counting each sample details as a row of the Tablei=0
i=0;
```

```
while i< ss

%---- Step 01: set the x,y of BS1
x_b1=0.0000; y_b1=0.0000; z_b1=0.0000;

%---- Step 02: Generate the x,y of Scatter-1
x_s1=4.0000*coef+ceil(rand()*20.0000*coef);
%y_s1=4.0000*coef+ceil(rand()*18.0000*coef);
y_s1=4.0000*coef+ceil(rand()*20.0000*round(coef/5));
z_s1=4.0000*coef+ceil(rand()*20.0000*coef);

%---- Step 03: Calculate r1 and r5
r1=sqrt((x_s1)^2+(y_s1)^2);
r5=sqrt((x_s1)^2+(z_s1)^2);

%---- Step 04: Calculate aod1, aod3
aod1=asind((y_s1/r1));
aod3=asind((z_s1/r5));

%if(((aod1<=35)&&(aod1>=75))||((aod3<=35)&&(aod3>=75))) continue
%end;

%---- Step05: Generate the x,y of Mobile Station
x_m=x_s1+4.0000*coef+ceil(rand()*(20.0000*coef));
%y_m=6.0000*coef+ceil(rand()*(y_s1-12.0000*coef));
%y_m=6.0000*coef+ceil(rand()*(y_s1-12.0000*round(coef/5)));
y_m=round(y_s1/4)+ceil(rand()*(y_s1*3/4));
z_m=round(z_s1/4)+ceil(rand()*(z_s1*3/4));

%---- Step 06: Calculate r2
r2=sqrt((x_m-x_s1)^2+(y_m-y_s1)^2);
r6=sqrt((x_m-x_s1)^2+(z_m-z_s1)^2);

%---- Step 07: Calculate aoa1, aoa3
```

```
aoa1=asind((y_s1-y_m)/r2);
aoa3=asind((z_s1-z_m)/r6);

%if(((aoa1<=35)&&(aoa1>=75))||((aoa3<=35)&&(aoa3>=75))) continue
%end;

%---- Step08: Generate the x,y of Scatter-2
x_s2=x_m+4.0000*coef+ceil(rand()*(20.0000*coef));
%y_s2=y_m+4.0000*coef+ceil(rand()*(y_m+16.0000*coef));
y_s2=y_m+4.0000*coef+ceil(rand()*(16.0000*round(coef/5)));
z_s2=z_m+4.0000*coef+ceil(rand()*(16.0000*coef));

%---- Step 09: Calculate r3, and r7
r3=sqrt((x_s2-x_m)^2+(y_s2-y_m)^2);
r7=sqrt((x_s2-x_m)^2+(z_s2-z_m)^2);

%---- Step 10: Calculate aoa2 and aoa4
aoa2=asind((y_s2-y_m)/r3);
aoa4=asind((z_s2-z_m)/r7);

%if(((aoa2<=35)&&(aoa2>=75))||((aoa4<=35)&&(aoa4>=75))) continue
%end;

%---- Step11: Generate the x,y of Base Station 2
x_b2=x_s2+4.0000*coef+ceil(rand()*(20.0000*coef));
y_b2=0.0000;
z_b2=0.0000;

%---- Step 12: Calculate r4 and r8
r4=sqrt((x_b2-x_s2)^2+(y_s2)^2);
r8=sqrt((x_b2-x_s2)^2+(z_s2)^2);

%---- Step 13: Calculate aod2 and aod4
aod2=asind((y_s2)/r4);
```

```
aod4=asind((z_s2)/r8);

%if(((aod2<=35)&&(aod2>=75))||((aod4<=35)&&(aod4>=75))) continue
%end;
%---- Step 14: Calculate dl1, dl2, dl3, and dl4
dl1= r1+r2;       dl2=r3+r4;
dl3= r5+r6;       dl4=r7+r8;

    x5=dl1*cosd(aod1);
    y5=dl1*sind(aod1);
    x6=x_b2-dl2*cosd(aod2);
    y6=dl2*sind(aod2);

    x7=dl3*cosd(aod3);
    z7=dl3*sind(aod3);

if((aod1<=a_min)||(aod1>=a_max)||(aod2<=a_min)||(aod2>=a_max)||(aod3<=a_min)
||(aod3>=a_max)||(aod4<=a_min)||(aod4>=a_max)||(aoa1<=a_min)||(aoa1>=a_max)||
(aoa2<=a_min)||(aoa2>=a_max)||(aoa3<=a_min)||(aoa3>=a_max)||(aoa4<=a_min)||(a
oa4>=a_max))
   %fprintf('\n aod1=%f, aoa1=%f, aod2=%f, aoa2=%f,',aod1,aoa1,aod2,aoa2);
   %fprintf('\n aod3=%f, aoa3=%f, aod4=%f, aoa4=%f ',aod3,aoa3,aod4,aoa4);
   %fprintf('\n SCT1(%f, %f, %f), MS(%f, %f, %f),', x_s1, y_s1, z_s1, x_m, y_m,
z_m);
   %fprintf('\n SCT2(%f, %f, %f), BS2(%f,0,0) ',x_s2, y_s2, z_s2, x_b2);
   %return;

   continue;
end;

if((x6>(x5+2.0000*coef))&&(aod1<90.000)&&(aoa1<90.0000)&&(aoa2<90.0000)
&&(aod2<90.0000)&&(aod1>0.0000)&&(aoa1>0.0000)&&(aoa2>0.0000)&&(aod2
>0.0))
%---- Print the table data row
```

```
fprintf(fw,'%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t',x_b1, y_b1, z_b1, r1, aod1);
fprintf(fw,'%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t',x_s1, y_s1, z_s1, r2, aoa1,
dl1);
fprintf(fw,'%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t',x_m, y_m, z_m, r3, aoa2);
fprintf(fw,'%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t',x_s2, y_s2, z_s2, r4, aod2,
dl2);
fprintf(fw,'%8.4f\t%8.4f\t%8.4f\t',x_b2, y_b2, z_b2);
fprintf(fw,'%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t',r5, r6, dl3, aod3, aoa3);
fprintf(fw,'%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f',r7, r8, dl4, aod4, aoa4);

if (i+1<ss) fprintf(fw,'\t\n');
end;

fprintf(fw2,'%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t',x_b1, y_b1, z_b1, r1, aod1);
fprintf(fw2,'%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t',x_s1, y_s1, z_s1, r2, aoa1,
dl1);
fprintf(fw2,'%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t',x_m, y_m, z_m, r3, aoa2);
fprintf(fw2,'%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t',x_s2, y_s2, z_s2, r4, aod2,
dl2);
fprintf(fw2,'%8.4f\t%8.4f\t%8.4f\t',x_b2, y_b2, z_b2);
fprintf(fw2,'%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t',r5, r6, dl3, aod3, aoa3);
fprintf(fw2,'%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f',r7, r8, dl4, aod4, aoa4);

if (i+1<ss)
   fprintf(fw2,'\t\n');
end;
fprintf('\n Sample:%d',i+1);
i=i+1;
end    %-- End of if

end
%---- End of while

fclose(fw);
```

```
fclose(fw2);
load gong
sound(y,Fs)
h = msgbox('Operation Completed----');
```

Appendix B: MATLAB Source Code of ACE-FARM Algorithm

```
% ACE 3D Positioning with 2 BSs

clc;

clear;

format long;

st='_Option_a';

%---- For Statistics ---------

t_err_x=0.0000;

t_err_y=0.0000;

t_err_z=0.0000;

t_cost=0.0000;

%--- Conditional turns -------

cnt2=0;

turn_allowed=100;

%---- Constants and parameters

k=11;

v=2^-k;

u=2^-(2*k+1);

err=10^-k;

total_cost=0;

m_err=0.500000;       %-- Accepteble error for distance;

test_step=1023;       %-- which sample will be traced

step_value=1.00000;

s_step_value='1.00000';
```

```
sample_date='20191226';

test_date='20200113B';

ver=strcat('-K',int2str(k),'-SV',s_step_value);

fr=fopen(strcat(pwd,'\','SampleSet-',sample_date,'-SS1000-d.txt'),'r');

fw=fopen(strcat(pwd,'\','TestResults',test_date,ver,'-SS1000-d.txt'),'w');

fprintf(fw,'%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\n',' xm',' ym',' zm',' cx1',' cy1',' cx2',' cy2',' cx8',' cz8',' x1',' y1',' x2',' y2','(xm-cx1)','(ym-cy1)','(zm-cz8)','(-COST-)');

cnt=0;

tot_cost=0;

cy=0;

%----------------------

while ~feof(fr)

cnt2=cnt2+1;

if(cnt2>turn_allowed)

   break;

end;

   %---- Constants and parameters

%k=7;

%v=2^-k;

%u=2^-(2*k+1);

%err=10^-k;

%step_value=5.0000;

cnt=cnt+1;
```

```
total_cost=0;
fprintf('\n\n\n<Sample-%3d is in process!>',cnt);

    xb1=fscanf(fr,'%f',1);
    yb1=fscanf(fr,'%f',1);
    zb1=fscanf(fr,'%f',1);

    r1=fscanf(fr,'%f',1);
    aod1=fscanf(fr,'%f',1);
    alphainp1= aod1;

    sx1=fscanf(fr,'%f',1);
    sy1=fscanf(fr,'%f',1);
    sz1=fscanf(fr,'%f',1);

    r2=fscanf(fr,'%f',1);
    aoa1=fscanf(fr,'%f',1);
    Dl1=fscanf(fr,'%f',1);

    xm=fscanf(fr,'%f',1);
    ym=fscanf(fr,'%f',1);
    zm=fscanf(fr,'%f',1);

    r3=fscanf(fr,'%f',1);
```

```
aoa2=fscanf(fr,'%f',1);


sx2=fscanf(fr,'%f',1);
sy2=fscanf(fr,'%f',1);
sz2=fscanf(fr,'%f',1);


r4=fscanf(fr,'%f',1);
aod2=fscanf(fr,'%f',1);
Dl2=fscanf(fr,'%f',1);
alphainp2=aod2;



xb2=fscanf(fr,'%f',1);
yb2=fscanf(fr,'%f',1);
zb2=fscanf(fr,'%f',1);


r5=fscanf(fr,'%f',1);
r6=fscanf(fr,'%f',1);
Dl3=fscanf(fr,'%f',1);
aod3=fscanf(fr,'%f',1);
aoa3=fscanf(fr,'%f',1);


r7=fscanf(fr,'%f',1);
r8=fscanf(fr,'%f',1);
```

```
    Dl4=fscanf(fr,'%f',1);
    aod4=fscanf(fr,'%f',1);
    aoa4=fscanf(fr,'%f',1);

fprintf('\n\nBS1(0,0,0)');
fprintf('\nSC1(%f,%f,%f)',sx1,sy1,sz1);
fprintf('\nMS(%f,%f,%f)',xm,ym,zm);
fprintf('\nSC2(%f,%f,%f)',sx2,sy2,sz2);
fprintf('\nBS2(%f,0,0)\n',xb2);
fprintf('\n\nDL1=%f,  aod1=%f,  aoa1=%f',Dl1,aod1,aoa1);
fprintf('\nDL2=%f,  aod2=%f,  aoa2=%f',Dl2,aod2,aoa2);
fprintf('\nDL3=%f,  aod3=%f,  aoa3=%f',Dl3,aod3,aoa3);
fprintf('\nDL4=%f,  aod4=%f,  aoa4=%f\n',Dl4,aod4,aoa4);
clf;
%--- Sağlama yapmak için
    x5=Dl1*cosd(aod1);
    y5=Dl1*sind(aod1);
    x6=xb2-Dl2*cosd(aod2);
    y6=Dl2*sind(aod2);

    x7=Dl3*cosd(aod3);
    z7=Dl3*sind(aod3);

    s1=(y5-ym)/(x5-xm);  %- s1: Slope of R1
```

```
    p1=y5-(s1*x5);
    e1=-p1/s1;          %- Where the x-axis crossed (e1,0)

    s2=(y6-ym)/(x6-xm);  %- s2: Slope of R4
    p2=y6-(s2*x6);
    e2=-p2/s2;          %- Where the x-axis crossed (e2,0)

%*****************************************************************
******
% BASE-1 SIDE
%*****************************************************************
******
%-----------------------------------------------------------------
% The DL1 lay on x-axis, and then rotated with angle rad_aod1
% x1, y1 are the coordinates of the vector head
rad_aod1=(aod1*pi)/180;
ro=0.0000;
x1=Dl1;
y1=0.0000;
if (rad_aod1>pi/2)
   sign=-1;
   delta=pi-rad_aod11;
else
   sign=1;
   delta=rad_aod1;
```

```
end


% Rotating DL-1 with angle aod1(delta) :
while((delta-ro)>=err)
                                        total_cost=total_cost+1;
 oldx1=x1;
 x1=x1-x1*u-y1*v;
 y1=y1-y1*u+oldx1*v;
 ro=ro+2^-k;
end %while
fprintf('\n\nRotating DL-1(%f,%f) \nwith angle rad_aod1(%f)',x1,y1,delta);
fprintf('\nCalculated arrow head position(%f,%f)',x5,y5);
fprintf('\nCost: %10d',total_cost);
%------------------------------------------------------------------------



%------------------------------------------------------------------------
% Stepper vector V11 lay on x-axis and then rotated with angle rad-aod1
% Stepper vector V11 will be used for decrease on R1 vector on vector break
% process of the vector DL-1
% x11,y11 are the coordinates of the vector head of the stepper vector V11
%x11=0.1000;
x11=step_value;
y11=0.0000;
```

```
ro=0.0000;
%alpha21=(alphainp1*pi)/180;
%if (alpha21>pi/2)
%    sign=-1;
%    delta=pi-alpha21;
%else
%    sign=1;
%    delta=alpha21;
%end
%alpha21 de aod1 dir

%delta is same as the rad-aod1

while((delta-ro)>=err)
                                    total_cost=total_cost+1;
 oldx11=x11;
 x11=x11-x11*u-y11*v;
 y11=y11-y11*u+oldx11*v;
 ro=ro+2^-k;
end %while
c_x11=step_value*cosd(aod1);
c_y11=step_value*sind(aod1);
fprintf('\n\nRotating V11(%f, %f)',x11,y11);
fprintf('\nCalculated V11=(%f, %f)',c_x11,c_y11);
```

```
fprintf('\nwith angle rad_aod1(%f)',delta);

fprintf('\nCost=%d',total_cost);



%-------------------------------------------------------------------------


%-------------------------------------------------------------------------
% Stepper vector V12 lay on x-axis and then rotated with angle rad-aoa1
% Stepper vector V12 will be used for increase on R2 vector on vector break
% process of the vector DL-1
% x12,y12 are the coordinates of the vector head of the stepper vector V11
%x12=0.1000;
x12=step_value;
y12=0.0000;
ro=0.0000;
rad_aoa1=(aoa1*pi)/180;
if (rad_aoa1>pi/2)
   sign=-1;
   delta=pi-rad_aoa1;
else
   sign=1;
   delta=rad_aoa1;
end
```

```
while((delta-ro)>=err)
                                    total_cost=total_cost+1;
oldx12=x12;
x12=x12-x12*u-y12*v;
y12=y12-y12*u+oldx12*v;
ro=ro+2^-k;
end %while
c_x12=step_value*cosd(aoa1);
c_y12=step_value*sind(aoa1);
fprintf('\n\nRotating V12(%f, %f)',x12,y12);
fprintf('\nCalculated V12=(%f ,%f)',c_x12,c_y12);
fprintf('\nwith angle rad_aoa1(%f)',delta);
fprintf('\nCost=%d',total_cost);
%-----------------------------------------------------------------------


%*************************************************************************
******
% BASE-2 SIDE
%*************************************************************************
******
%-----------------------------------------------------------------------
% The DL2 lay on x-axis, and then rotated with angle rad_aod2
% x2, y2 are the coordinates of the vector head
rad_aod2=(aod2*pi)/180;
ro=0.0000;
```

```
x2=Dl2;
y2=0.0000;
if (rad_aod2>pi/2)
   sign=-1;
   delta=pi-rad_aod2;
else
   sign=1;
   delta=rad_aod2;
end

while((delta-ro)>=err)
                                    total_cost=total_cost+1;
  oldx2=x2;
  x2=x2-x2*u-y2*v;
  y2=y2-y2*u+oldx2*v;
  ro=ro+2^-k;
  x2_rev=xb2-x2;
end %while
x2=x2_rev;
% The vector head and vector base has reversed and base carried on to xb2
fprintf('\n\nRotating DL-2(%f,%f) \nwith angle rad_aod2(%f)',x2,y2,delta);
fprintf('\nCalculated arrow head position(%f,%f)',x6,y6);
fprintf('\nCost: %10d',total_cost);
```

```
%-------------------------------------------------------------------------

% Stepper vector V21 lay on x-axis and then rotated with angle rad-aod2

% Stepper vector V21 will be used for decrease on R4 vector on vector break

% process of the vector DL-2

% x21,y21 are the coordinates of the vector head of the stepper vector V21


%x21=0.1000;

x21=step_value;

y21=0.0000;

ro=0.0000;


%delta is same as rad_aod2

while((delta-ro)>=err)

                                        total_cost=total_cost+1;

 oldx21=x21;

 x21=x21-x21*u-y21*v;

 y21=y21-y21*u+oldx21*v;

 ro=ro+2^-k;

end %while

c_x21=step_value*cosd(aod2);

c_y21=step_value*sind(aod2);

fprintf('\n\nRotating V21(%f, %f)',x21,y21);

fprintf('\nCalculated V21=(%f ,%f)',c_x21,c_y21);

fprintf('\nwith angle rad_aod2(%f)',delta);
```

```
fprintf('\nCost=%d',total_cost);



%-----------------------------------------------------------------------
% Stepper vector V22 lay on x-axis and then rotated with angle rad-aoa2
% Stepper vector V22 will be used for an increase on R3 vector on vector break
% process of the vector DL-2
% x22,y22 are the coordinates of the vector head of the stepper vector V22
%x22=0.1000;
x22=step_value;
y22=0.0000;
ro=0.0000;

rad_aoa2=(aoa2*pi)/180;
if (rad_aoa2>pi/2)
   sign=-1;
   delta=pi-rad_aoa2;
else
   sign=1;
   delta=rad_aoa2;
end

while((delta-ro)>=err)
                        total_cost=total_cost+1;
```

```
 oldx22=x22;
 x22=x22-x22*u-y22*v;
 y22=y22-y22*u+oldx22*v;
 ro=ro+2^-k;
end %while
c_x22=step_value*cosd(aoa2);
c_y22=step_value*sind(aoa2);
fprintf('\n\nRotating V22(%f, %f)',x22,y22);
fprintf('\nCalculated V22=(%f ,%f)',c_x22,c_y22);
fprintf('\nwith angle rad_aoa2(%f)',delta);
fprintf('\nCost=%d',total_cost);

%-------------------------------------------------------------------

%*************************************************************************
*******
% Vectors DL-1 and Dl-2 will be broken until y1 == y2
%*************************************************************************
*******
 cx1=x1;     % Head-x of R2
 cy1=y1;     % Head-y of R2
 cx2=x2;     % Head-x of R3
 cy2=y2;     % Head-y of R3
 xtemp=0;    % length-x of R2
 ytemp=0;    % length-y of R2
```

```
 x2temp=0;   % length-x of R3
 y2temp=0;   % length-y of R3
fprintf('\n\nCompare T1(%f,%f) & T2(%f,%f)',x1,y1,x2,y2);

if (y1>y2)
fprintf('\nT1(y)>T2(y), so Break DL-1 until T(y1)==T(y2)');
fprintf('\nError=%f \n',err);

 while ((cy1-y2)>err)
% while ((cy1-y2)>m_err)
                        total_cost=total_cost+1;
    x1=x1-x11;
    y1=y1-y11;                  % reduce stepper vector V11 from R1
    h_xy1= sqrt(x1^2+y1^2);         % current R1 = sqrt(x1^2+y1^2);
                        total_cost=total_cost+113;
    while((Dl1-(h_xy1 +sqrt(xtemp^2+ytemp^2)))>err)
%    while((Dl1-(h_xy1 +sqrt(xtemp^2+ytemp^2)))>(m_err/2))
                        % add stepper vector V12 ro R2
                        % until (R1+R2) == DL1
                        % current R2 = sqrt(xtemp^2+ytemp^2);
                        total_cost=total_cost+113;
      xtemp=xtemp+x12;
      ytemp=ytemp+y12;
      cx1=x1+xtemp;
```

```
      cy1=y1-ytemp;
                              total_cost=total_cost+4;
     end
  end %while((cy1-y2)>err)
fprintf('\nNow, DL1(%f, %f)== DL2(%f, %f): %10d',cx1,cy1,cx2,cy2);
fprintf('\nCost=%f\n',total_cost);

else %y2>y1
fprintf('\nT2(y)>T1(y), so Break DL-2 until T(y2)==T(y1)');
fprintf('\nError=%f \n',err);

 while((cy2-cy1)>err)
 %while((cy2-cy1)>m_err)
   x2=x2+x21;
   y2=y2-y21;                  % reduce stepper vector V21 from R4
                              total_cost=total_cost+5;
   h_xy2= sqrt((x2-xb2)^2+y2^2); % current R4= sqrt((x2-xb2)^2+y2^2);
                              total_cost=total_cost+112;
   while((Dl2-(h_xy2 +sqrt(x2temp^2+y2temp^2)))>err)
  %while((Dl2-(h_xy2 +sqrt(x2temp^2+y2temp^2)))>(m_err/2))
                              % add stepper vector V22 to R3
                              % until (R3+R4) == DL2
                              % current R3 = sqrt(x2temp^2+y2temp^2);
                              total_cost=total_cost+113;
```

```
     x2temp=x2temp+x22;
     y2temp=y2temp+y22;
     cx2=x2-x2temp;
     cy2=y2-y2temp;
                              total_cost=total_cost+4;
    end
   end %while((y2-y1)>err)
fprintf('\nNow, DL1(%f, %f)== DL2(%f, %f): %10d',cx1,cy1,cx2,cy2);
fprintf('\nCost=%f\n',total_cost);
end % (y1>y2)

%-------------------------------------------------------------
%
% Compare AOA angles and select the greatest to put it into the
% inner loop - greatest angles yield smaller x-axis moves.
%
%-------------------------------------------------------------
fprintf('\n\nCompare aoa1(%f) & aoa2(%f)',aoa1,aoa2);
%-------------------------------------------------------------

if (aoa1>aoa2)
%if (aoa1<aoa2)
    fprintf('\naoa1 > aoa2, First R3 will be broken');
    fprintf('\nThen R4 will be carried to near R3');
```

```
   fprintf('\nA(%f, %f), P1(%f,%f), C(%f, %f),
P2(%f,%f)',x1,y1,cx1,cy1,x2,y2,cx2,cy2);
   while ((cx2-cx1)>err)
%    while ((cx2-cx1)>m_err)
%---- Break the DL2, one step ----------------------------------------
  x2=x2+x21;
  y2=y2-y21;            % reduce stepper vector V21 from R4
                    total_cost=total_cost+3;
  h_xy3 = sqrt((x2-xb2)^2+y2^2);
                    % current R4= sqrt((x2-xb2)^2+y2^2);
                    total_cost=total_cost+114;
  while((Dl2-(h_xy3 +sqrt(x2temp^2+y2temp^2)))>err)
%  while((Dl2-(h_xy3 +sqrt(x2temp^2+y2temp^2)))>(m_err))
                    % add stepper vector V22 to R3
                    % until (R3+R4) == DL2
                    % current R3 = sqrt(x2temp^2+y2temp^2);
                    total_cost=total_cost+113;
    x2temp=x2temp+x22;
    y2temp=y2temp+y22;
    cx2=x2-x2temp;
    cy2=y2-y2temp;
                              total_cost=total_cost+4;
  end
%---- Breake DL1 several steps until cy1==cy2 -------------------------
   while((cy1-cy2)>err)
```

```
%    while(abs(cy1-cy2)>m_err)
if (cnt2==test_step)
    fprintf('\n\t\t\t\t\t C(%f, %f), P2(%f,%f)\n',x2,y2,cx2,cy2);
    %fprintf('\n  m_err=%f',m_err);
    fprintf('\n  xtemp/ytemp(%f, %f) - x2t/y2t(%f, %f)', xtemp, ytemp, x2temp,
y2temp);
    fprintf('\n  DL1-G(%f), DL1-C(%f)',Dl1,
sqrt(x1^2+y1^2)+sqrt(xtemp^2+ytemp^2));
    fprintf('\n  DL2-G(%f), DL2-C(%f)',Dl2,sqrt((x2-
xb2)^2+y2^2)+sqrt(x2temp^2+y2temp^2));

end;
     x1=x1-x11;
     y1=y1-y11;          % reduce stepper vector V11 from R1
                    total_cost=total_cost+2;
     h_xy4= sqrt(x1^2+y1^2);
                    % current R1= sqrt(x1^2+y1^2);
                    total_cost=total_cost+113;
    if(x1<0)
       fprintf('\n Error x1(%f)<0',x1);
       return;
    end;

    if((Dl1-(h_xy4 +sqrt(xtemp^2+ytemp^2)))<err)
       fprintf('\n Errordan küçük ---> line 475');
```

```
        continue;
    end;
     while((Dl1-(h_xy4 +sqrt(xtemp^2+ytemp^2)))>err)
  %   while((Dl1-(h_xy4 +sqrt(xtemp^2+ytemp^2)))>(m_err/4))
                    % add stepper vector V12 to R2
                    % until (R1+R2) == DL1
                    % current R2 = sqrt(xtemp^2+ytemp^2);
                            total_cost=total_cost+115;
      xtemp=xtemp+x12;
      ytemp=ytemp+y12;
      cx1=x1+xtemp;
      cy1=y1-ytemp;
                            total_cost=total_cost+4;



if((cx1-cx2)>err)
  fprintf('\n(Err)P1(%f,%f) & P2(%f,%f)',cx1,cy1,cx2,cy2);
end
     end %while(DL1!=R1+R2) extend R2
    if(cnt2==test_step)
    fprintf('\n...R1=%f,',h_xy5);
    fprintf('\nA(%f, %f), P1(%f,%f)\n',x1,y1,cx1,cy1);
    fprintf('\n  xtemp/ytemp(%f, %f) - x2t/y2t(%f, %f)', xtemp, ytemp, x2temp,
y2temp);
```

```
    fprintf('\n  DL1-G(%f), DL1-C(%f)',Dl1,
sqrt(x1^2+y1^2)+sqrt(xtemp^2+ytemp^2));
    fprintf('\n  DL2-G(%f), DL2-C(%f)',Dl2,sqrt((x2-
xb2)^2+y2^2)+sqrt(x2temp^2+y2temp^2));
     pause;
    end;


   end %while((cy1-cy2)>err)
 end %while((cx2-cx1)>err)
fprintf('\n\nMS(%f,%f),MS(%f,%f) Found!(aoa1<aoa2)',cx1,cy1,cx2,cy2);
fprintf('\nMS(%f,%f) Given',xm,ym);
fprintf('\nCost=%10d',total_cost);


else %aoa2>aoa1
%-------------------------------------------------------------
%
%-- Break the DL1, one step ----------------------------------
%
%-------------------------------------------------------------
fprintf('\naoa2 > aoa1, Smaller angle gives larger x, smaller y moves');
fprintf('\nSo, First R1 will be broken');
fprintf('\nThen R2 will be carried to near R1');
fprintf('\nP1(%f,%f), P2(%f,%f)\n',cx1,cy1,cx2,cy2)
while ((cx2-cx1)>err)
%while ((cx2-cx1)>m_err)
```

```
% reduce stepper vector V11 from R1
if(cnt2==test_step)
    fprintf('\nP1(%f, %f) - P2(%f,%f)',cx1,cy1,cx2,cy2);
    fprintf('\n  m_err=%f m..',m_err);
    fprintf('\n  xt/yt(%f, %f) - xt2/yt2(%f, %f)', xtemp, ytemp, x2temp, y2temp);
    fprintf('\n  DL1-G(%f), DL1-C(%f)',Dl1,
sqrt(x1^2+y1^2)+sqrt(xtemp^2+ytemp^2));
    fprintf('\n  DL2-G(%f), DL2-C(%f)\n',Dl2,sqrt((x2-
xb2)^2+y2^2)+sqrt(x2temp^2+y2temp^2));
    h_xy5 = sqrt(x1^2+y1^2);
    fprintf('\n h_xy5=%f,',h_xy5);
    fprintf('\n Dl1-(h_xy5+R3)=%f,',(Dl1-(h_xy5 +sqrt(xtemp^2+ytemp^2))));
    pause;
end;
    x1=x1-x11;
    y1=y1-y11;
                        total_cost=total_cost+2;
    h_xy5 = sqrt(x1^2+y1^2);
                        % current R1= sqrt(x1^2+y1^2);
                        total_cost=total_cost+113;

    if(cnt2==test_step)
       fprintf('\n...x1=%f, y1=%f, h_xy5=%f,',x1,y1,h_xy5);
    end;
```

```
    if((Dl1-(h_xy5 +sqrt(xtemp^2+ytemp^2)))<err)
       fprintf('\n --> error : 543');
       return;
    end;

    while((Dl1-(h_xy5 +sqrt(xtemp^2+ytemp^2)))>err)
    %while((Dl1-(h_xy5 +sqrt(xtemp^2+ytemp^2)))>m_err)
                          % add stepper vector V12 to R2
                          % until (R1+R2) == DL1
                          % current R2 = sqrt(xtemp^2+ytemp^2);
                          total_cost=total_cost+113;
      xtemp=xtemp+x12;
      ytemp=ytemp+y12;
      cx1=x1+xtemp;
      cy1=y1-ytemp;
                               total_cost=total_cost+4;
    if(cnt2==test_step)
       fprintf('\n...x1=%f, y1=%f, h_xy5=%f,',x1,y1,h_xy5);
    end;
   end; %while (Dl1-(....

%---- Breake DL2 several steps until cy1==cy2 -------------------------
  while((cy2-cy1)>err)
  %while((cy2-cy1)>m_err)
```

```
% reduce stepper vector V21 from R4
   x2=x2+x21;
   y2=y2-y21;
                        total_cost=total_cost+2;
   h_xy6 = sqrt((x2-xb2)^2+y2^2);
   % current R4= sqrt((x2-xb2)^2+y2^2);
                        total_cost=total_cost+114;
% while((Dl2-( h_xy6 +sqrt(x2temp^2+y2temp^2)))>err)
if ((Dl2-( h_xy6 +sqrt(x2temp^2+y2temp^2)))<err)
   continue;
end
  while((Dl2-( h_xy6 +sqrt(x2temp^2+y2temp^2)))>err)
                            % add stepper vector V22 to R3
                            % until (R3+R4) == DL2
                            % current R3 = sqrt(x2temp^2+y2temp^2);
                               total_cost=total_cost+113;

    x2temp=x2temp+x22;
    y2temp=y2temp+y22;
    cx2=x2-x2temp; %xb2 hatası olduğunu düşünürüm.
    cy2=y2-y2temp;
                               total_cost=total_cost+4;
    %fprintf('+ %f +',cx2);
  end %while
```

```
  end %while((y2-y1)>err)
%2
 end %while((x2-x1)>err)
fprintf('\n\nMS(%f,%f),MS(%f,%f) Found!(aoa1<aoa2)',cx1,cy1,cx2,cy2);
fprintf('\nMS(%f,%f) Given',xm,ym);
fprintf('\nCost=%10d',total_cost);
end % (aoa1>aoa2)

%return;

% ------------------------------------------------------------------------
% Z-Axis Processes
% ------------------------------------------------------------------------
% Step 1: Vector angle rotation, finding the (x,z) coordinates with
%         angle AOD3, AOA3 and length DL3
% initial inputs

%step_value=step_value*0.1000;
rad_aod3=(aod3*pi)/180;
ro=0.0000;
x8=Dl3;
z8=0.0000;
if (rad_aod3>pi/2)
   sign=-1;
```

```
    delta=pi-rad_aod3;
else
    sign=1;
    delta=rad_aod3;
end

while((delta-ro)>err)
  oldx8=x8;
  x8=x8-x8*u-z8*v;
  z8=z8-z8*u+oldx8*v;
  ro=ro+2^-k;
end %while
fprintf('\n\nRotating DL-3(%f,%f)',x8,z8);
fprintf('\nwith angle rad_aod3(%f)',delta);
fprintf('\nCalculated arrow head position(%f,%f)',x7,z7);
fprintf('\nCost: %10d',total_cost);

%------------------------------------------------------------------------
% Stepper vector V81 lay on x-axis and then rotated with angle rad-aod3
% Stepper vector V81 will be used for decrease on R5 vector on vector break
% process of the vector DL-1
% x81,y81 are the coordinates of the vector head of the stepper vector V81

x81=step_value;
```

```
z81=0.0000;
ro=0.0000;
rad_aod3=(aod3*pi)/180;
if (rad_aod3>pi/2)
    sign=-1;
    delta=pi-rad_aod3;
else
    sign=1;
    delta=rad_aod3;
end

while((delta-ro)>=err)
  oldx81=x81;
  x81=x81-x81*u-z81*v;
  z81=z81-z81*u+oldx81*v;
  ro=ro+2^-k;
                                total_cost=total_cost+1;
end %while
c_x81=step_value*cosd(aod3);
c_z81=step_value*sind(aod3);
fprintf('\n\nRotating V81(%f, %f)',x81,z81);
fprintf('\nCalculated V81=(%f, %f)',c_x81,c_z81);
fprintf('\nwith angle rad_aod3(%f)',delta);
fprintf('\nCost=%d',total_cost);
```

```
%-------------------------------------------------------------------------


%-------------------------------------------------------------------------
% Stepper vector V82 lay on x-axis and then rotated with angle rad-aoa3
% Stepper vector V82 will be used for increase on R6 vector on vector break
% process of the vector DL-3
% x82,y82 are the coordinates of the vector head of the stepper vector V82


x82=step_value;
z82=0.0000;
ro=0.0000;
rad_aoa3=(aoa3*pi)/180;
if (rad_aoa3>pi/2)
   sign=-1;
   delta=pi-rad_aoa3;
else
   sign=1;
   delta=rad_aoa3;
end


while((delta-ro)>=err)
  oldx82=x82;
  x82=x82-x82*u-z82*v;
  z82=z82-z82*u+oldx82*v;
```

```
 ro=ro+2^-k;
                                total_cost=total_cost+1;
end %while
c_x82=step_value*cosd(aoa3);
c_z82=step_value*sind(aoa3);
fprintf('\n\nRotating V82(%f, %f)',x82,z82);
fprintf('\nCalculated V82=(%f, %f)',c_x82,c_z82);
fprintf('\nwith angle rad_aoa3(%f)',delta);
fprintf('\nCost=%d',total_cost);

%-------------------------------------------------------------------
%*********************************************************************
*******
% Vectors DL-3 will be broken until cx1 == cx8
%*********************************************************************
*******
fprintf('\n\nVector DL-3 will be broken untill cx8==cx1');
 cx8=x8;  % Head-x of R6
 cz8=z8;  % Head-z of R6
 xtemp=0; % length-x of R6
 ztemp=0; % length-z of R6

% m_err=m_err/2;
fprintf('\nCompare P1(%f, %f) & P3(%f, %f)',cx1,cy1,cx8,cz8);
 if(cx1>cx8)
```

```
    fprintf('\nP1x(%f) > P3x(%f)',cx1,cx8);
cnt3=0;
%  while ((cx1-cx8)>err)
  while ((cx1-cx8)>err)
    % reduce stepper vector V81 from R5
    x8=x8-x81;
    z8=z8-z81;
                                total_cost=total_cost+3;
    h_xy7 = sqrt(x8^2+z8^2);
          % current R5 = sqrt(x8^2+z8^2);
                                total_cost=total_cost+113;
    while((Dl3-(h_xy7 +sqrt(xtemp^2+ztemp^2)))>err)
  %  while((Dl3-(h_xy7 +sqrt(xtemp^2+ztemp^2)))>m_err)
                                    % add stepper vector V82 ro R6
                               % until (R5+R6) == DL3
                               % current R6 = sqrt(xtemp^2+ztemp^2);
                                total_cost=total_cost+113;
      xtemp=xtemp+x82;
      ztemp=ztemp+z82;
      cx8=x8+xtemp;
      cz8=z8-ztemp;
                                total_cost=total_cost+4;
    end % while((Dl3-(h_xy7 +sqrt(xtemp^2+ztemp^2)))>err)
```

```
 end %while((cx1-cx8)>err)
%    fprintf('\n(*)Cost> Found z in (cx1>cx8) sitution: %10d',total_cost);
fprintf('\n\nMS(x,y,z) FOUND for SAMPLE-%d (cx8<cx1)',cnt);
fprintf('\nMS-1(%f, %f, %f)',cx1,cy1,cz8);
fprintf('\nMS-2(%f, %f, %f)',cx2,cy2,cz8);
fprintf('\nMS(%f, %f, %f) Given',xm,ym,zm);
fprintf('\nCost=%10d',total_cost);

 else
    fprintf('\n P3x(%f) > P1x(%f)',cx8,cx1);
    %fprintf('\n(***)(cx1<=cx8) sitution');
    while ((cx8-cx1)>err)
    %while ((cx8-cx1)>m_err)
    x8=x8-x81;
    z8=z8-z81;
                            total_cost=total_cost+3;
    h_xy8 = sqrt(x8^2+z8^2);
                            total_cost=total_cost+111;


    while((Dl3-(h_xy8 +sqrt(xtemp^2+ztemp^2)))>err)
%     while((Dl3-(h_xy8 +sqrt(xtemp^2+ztemp^2)))>m_err)
                            total_cost=total_cost+113;
      xtemp=xtemp+x82;
```

```
       ztemp=ztemp+z82;
       cx8=x8+xtemp;
       cz8=z8-ztemp;
                              total_cost=total_cost+4;
     end
  end %while((cx1-cx8)>err)
%fprintf('\n(*)Cost> Finding z in (cx1<cx8) sitution: %10d',total_cost);
fprintf('\n\nMS(x,y,z) FOUND for SAMPLE-%d (cx1<cx8)',cnt);
fprintf('\nMS-1(%f, %f, %f)',cx1,cy1,cz8);
fprintf('\nMS-2(%f, %f, %f)',cx2,cy2,cz8);
fprintf('\nMS(%f, %f, %f) Given',xm,ym,zm);
fprintf('\nCost=%10d',total_cost);

 end



if (cy1>cy2)
    cy=cy2;
else
    cy=cy1;
end;
cy=(cy1+cy2)/2;
%cy=cy1;
%SON ADIMLAR SONU
```

```
t_err_x=t_err_x+ abs(xm-cx1);

t_err_y=t_err_y+ abs(ym-cy);

t_err_z=t_err_z+abs(zm-cz8);

t_cost=t_cost+total_cost;

fprintf(fw,'%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t
%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8d\n',xm,ym,zm,cx1,cy1,cx2,cy2,cx8,
cz8,x1,y1,x2,y2,(xm-cx1),(ym-cy),(zm-cz8), total_cost );

end  %while

%end %for

fprintf(fw,'\n k=%d, m_err=%f, ste_V=%f',k, m_err, step_value);

fprintf(fw,'\n Average Errors: mx-cx: %f, my-cy=%f, mz-cz:%f, Avg-Cost: %f
\n',t_err_x/cnt, t_err_y/cnt, t_err_z/cnt, t_cost/cnt);

fprintf('\n(*)Operation Completed\n\n\n');

fclose(fr);

fclose(fw);

load gong

sound(y,Fs)

%hold off

h = msgbox('Operation Completed');


%msgbox(strcat('x11:',num2str(x11),' y11:',num2str(y11),' x12:',num2str(x12),'
y12:',num2str(y12)));

%plot(x1,y1,'Color', [0.0, 1.0, 0.0], 'Marker', '+ ');
```

Appendix C: MATLAB Source Code of ACE-DARM Algorithm

```
% ACE-DARM 3D Positioning with 2 BSs

clc;

clear;

format long;

st='_Option_a';

%---- For Statistics ---------

t_err_x=0.0000;

t_err_y=0.0000;

t_err_z=0.0000;

t_cost=0.0000;

%--- Conditional turns -------

cnt2=0;

turn_allowed=100;

k_start=7;

k_rotate=11;

%---- Constants and parameters

k=6;

v=2^-k;

u=2^-(2*k+1);

err=10^-k;

total_cost=0;

m_err=0.500000;       %-- Accepteble error for distance;

test_step=1024;       %-- which sample will be traced

step_value=1.00000;
```

```
s_step_value='1.00000';
sample_date='20191226';
test_date='20201001';
ver=strcat('-K',int2str(k),'-SV',s_step_value);
fr=fopen(strcat(pwd,'\','SampleSet-',sample_date,'-SS1000-d.txt'),'r');
fw=fopen(strcat(pwd,'\','TestResults_dynamic_',test_date,ver,'-SS1000-d.txt'),'w');
fprintf(fw,'%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t%8s\t
%8s\t%8s\t%8s\t%8s\n',' xm',' ym',' zm',' cx1',' cy1',' cx2',' cy2','
cx8',' cz8',' x1',' y1',' x2',' y2','(xm-cx1)','(ym-cy1)','(zm-cz8)','(-COST-
)');
cnt=0;
tot_cost=0;
cy=0;
%----------------------
while ~feof(fr)
cnt2=cnt2+1;
if(cnt2>turn_allowed)
   break;
end;
cnt=cnt+1;
total_cost=0;
fprintf('\n\n\n<Sample-%3d is in process!>',cnt);

    xb1=fscanf(fr,'%f',1);
    yb1=fscanf(fr,'%f',1);
```

```
zb1=fscanf(fr,'%f',1);

r1=fscanf(fr,'%f',1);
aod1=fscanf(fr,'%f',1);
alphainp1= aod1;

sx1=fscanf(fr,'%f',1);
sy1=fscanf(fr,'%f',1);
sz1=fscanf(fr,'%f',1);

r2=fscanf(fr,'%f',1);
aoa1=fscanf(fr,'%f',1);
Dl1=fscanf(fr,'%f',1);

xm=fscanf(fr,'%f',1);
ym=fscanf(fr,'%f',1);
zm=fscanf(fr,'%f',1);

r3=fscanf(fr,'%f',1);
aoa2=fscanf(fr,'%f',1);

sx2=fscanf(fr,'%f',1);
sy2=fscanf(fr,'%f',1);
sz2=fscanf(fr,'%f',1);
```

```
    r4=fscanf(fr,'%f',1);
    aod2=fscanf(fr,'%f',1);
    Dl2=fscanf(fr,'%f',1);
    alphainp2=aod2;


    xb2=fscanf(fr,'%f',1);
    yb2=fscanf(fr,'%f',1);
    zb2=fscanf(fr,'%f',1);

    r5=fscanf(fr,'%f',1);
    r6=fscanf(fr,'%f',1);
    Dl3=fscanf(fr,'%f',1);
    aod3=fscanf(fr,'%f',1);
    aoa3=fscanf(fr,'%f',1);

    r7=fscanf(fr,'%f',1);
    r8=fscanf(fr,'%f',1);
    Dl4=fscanf(fr,'%f',1);
    aod4=fscanf(fr,'%f',1);
    aoa4=fscanf(fr,'%f',1);

fprintf('\n\nBS1(0,0,0)');
fprintf('\nSC1(%f,%f,%f)',sx1,sy1,sz1);
fprintf('\nMS(%f,%f,%f)',xm,ym,zm);
```

```
fprintf('\nSC2(%f,%f,%f)',sx2,sy2,sz2);
fprintf('\nBS2(%f,0,0)\n',xb2);
fprintf('\n\nDL1=%f,  aod1=%f,  aoa1=%f',Dl1,aod1,aoa1);
fprintf('\nDL2=%f,  aod2=%f,  aoa2=%f',Dl2,aod2,aoa2);
fprintf('\nDL3=%f,  aod3=%f,  aoa3=%f',Dl3,aod3,aoa3);
fprintf('\nDL4=%f,  aod4=%f,  aoa4=%f\n',Dl4,aod4,aoa4);
clf;
%--- Sağlama yapmak için
    x5=Dl1*cosd(aod1);
    y5=Dl1*sind(aod1);
    x6=xb2-Dl2*cosd(aod2);
    y6=Dl2*sind(aod2);

    x7=Dl3*cosd(aod3);
    z7=Dl3*sind(aod3);

    s1=(y5-ym)/(x5-xm);  %- s1: Slope of R1
    p1=y5-(s1*x5);
    e1=-p1/s1;          %- Where the x-axis crossed (e1,0)

    s2=(y6-ym)/(x6-xm);  %- s2: Slope of R4
    p2=y6-(s2*x6);
    e2=-p2/s2;          %- Where the x-axis crossed (e2,0)

%*****************************************************************
```

```
% BASE-1 SIDE
%*****************************************************************
%-----------------------------------------------------------------
% The DL1 lay on x-axis, and then rotated with angle rad_aod1
% x1, y1 are the coordinates of the vector head
rad_aod1=(aod1*pi)/180;
ro=0.00000000000;
x1=Dl1;
y1=0.0000;
if (rad_aod1>pi/2)
   sign=-1;
   delta=pi-rad_aod11;
else
   sign=1;
   delta=rad_aod1;
end

% Rotating DL-1 with angle aod1(delta) :
%-------------------------- Loop k=4 to 11 -----
% err = err=10^-k;
%while((delta-ro)>=err)

fprintf('\nRotating DL1::');
 k=k_rotate;
 v=2^-k;
```

```
u=2^-(2*k+1);


while((delta-ro)>=(10^-k))
                                total_cost=total_cost+1;
fprintf('\n#k=%d,  (%f, %f)=>(%f, %f), delta=%f ro=%f, fark=%f; error=%f', k, x1,
y1, x5, y5, delta, ro, (delta-ro),(10^-k));
p_x1=x1;
p_y1=y1;


oldx1=x1;
x1=x1-x1*u-y1*v;
y1=y1-y1*u+oldx1*v;
ro=ro+2^-k;



while (abs(delta-ro)<(10^-k))
  if (k<11)
    k=k+1;
                                  total_cost=total_cost+1;
    %fprintf('\n ==>k++=%d;', k);
    %fprintf('delta=%f ro=%f, fark=%13.11f; error=%13.11f', delta, ro, (delta-
ro),(10^-k));
  end %if (k<11)


  if(k==11)
```

```
      break;
   end %if(k==11)


   if((delta-ro)<0)
      ro=ro-2^-k;
      x1=p_x1;
      y1=p_y1;
      total_cost=total_cost-1;
   end  %if((delta-ro)<0)



 end %while ((delta-ro)<(10^-k))
 %fprintf('#');


    if(x1<0)
    return;
   end %if (x1<0)


end %while((delta-ro)>=10^-k)


fprintf('\n\nRotated DL-1(%f,%f) \nwith angle rad_aod1(%f)',x1,y1,delta);
fprintf('\nCalculated arrow head position(%f,%f)',x5,y5);
fprintf('\nError: %12.11f',(delta-ro));
fprintf('\nCost: %10d',total_cost);
%----------------------------------------------------------------------
```

```
%-------------------------------------------------------------------------
% Stepper vector V11 lay on x-axis and then rotated with angle rad-aod1
% Stepper vector V11 will be used for decrease on R1 vector on vector break
% process of the vector DL-1
% x11,y11 are the coordinates of the vector head of the stepper vector V11
x11=step_value;
y11=0.0000;
ro=0.0000;

%alpha21 is aod1
%delta is same as the rad-aod1
k=k_rotate;
v=2^-k;
u=2^-(2*k+1);

while((delta-ro)>=10^-k)
                        total_cost=total_cost+1;
 oldx11=x11;
 x11=x11-x11*u-y11*v;
 y11=y11-y11*u+oldx11*v;
 ro=ro+2^-k;
  while (abs(delta-ro)<10^-k)
   if (k<11)
      k=k+1;
                          total_cost=total_cost+1;
```

```
  end %if (k<11)
  if(k==11)
     break;
  end %if(k==11)
  if((delta-ro)<0)
     ro=ro-2^-k;
     total_cost=total_cost-1;
  end %if((delta-ro)<0)
 end %while ((delta-ro)<10^-k)
end %while((delta-ro)>=10^-k)

c_x11=step_value*cosd(aod1);
c_y11=step_value*sind(aod1);
fprintf('\n\nRotating V11(%f, %f)',x11,y11);
fprintf('\nCalculated V11=(%f, %f)',c_x11,c_y11);
fprintf('\nwith angle rad_aod1(%f)',delta);
fprintf('\nCost=%d',total_cost);



%-------------------------------------------------------------------------
% Stepper vector V12 lay on x-axis and then rotated with angle rad-aoa1
% Stepper vector V12 will be used for increase on R2 vector on vector break
% process of the vector DL-1
% x12,y12 are the coordinates of the vector head of the stepper vector V11
```

```
x12=step_value;
y12=0.0000;
ro=0.0000;
rad_aoa1=(aoa1*pi)/180;
if (rad_aoa1>pi/2)
    sign=-1;
    delta=pi-rad_aoa1;
else
    sign=1;
    delta=rad_aoa1;
end %if (rad_aoa1>pi/2)

k=k_rotate;
v=2^-k;
u=2^-(2*k+1);
while((delta-ro)>=(10^-k))
                        total_cost=total_cost+1;
oldx12=x12;
x12=x12-x12*u-y12*v;
y12=y12-y12*u+oldx12*v;
ro=ro+2^-k;
 while (abs(delta-ro)<10^-k)
   if (k<11)
     k=k+1;
                            total_cost=total_cost+1;
```

```
  end %if (k<11)
  if(k==11)
     break;
  end %if(k==11)
  if((delta-ro)<0)
     ro=ro-2^-k;
     total_cost=total_cost-1;
  end %if((delta-ro)<0)
 end %while ((delta-ro)<10^-k)
end %while((delta-ro)>=10^-k)

c_x12=step_value*cosd(aoa1);
c_y12=step_value*sind(aoa1);
fprintf('\n\nRotating V12(%f, %f)',x12,y12);
fprintf('\nCalculated V12=(%f ,%f)',c_x12,c_y12);
fprintf('\nwith angle rad_aoa1(%f)',delta);
fprintf('\nCost=%d',total_cost);
%-----------------------------------------------------------------------

%***********************************************************************
% BASE-2 SIDE
%***********************************************************************
%-----------------------------------------------------------------------
% The DL2 lay on x-axis, and then rotated with angle rad_aod2
% x2, y2 are the coordinates of the vector head
```

```
rad_aod2=(aod2*pi)/180.00000000000;
ro=0.00000000000;
x2=Dl2;
y2=0.0000;
if (rad_aod2>pi/2)
    sign=-1;
    delta=pi-rad_aod2;
else
    sign=1;
    delta=rad_aod2;
end %if (rad_aod2>pi/2)

k=k_rotate;
v=2^-k;
u=2^-(2*k+1);
while((delta-ro)>=(10^-k))
                              total_cost=total_cost+1;
  oldx2=x2;
  x2=x2-x2*u-y2*v;
  y2=y2-y2*u+oldx2*v;
  ro=ro+2^-k;
 % x2_rev=xb2-x2;
  while (abs(delta-ro)<(10^-k))
   if (k<11)
     k=k+1;
```

```
            total_cost=total_cost+1;
   end %if (k<11)
   if(k==11)
      break;
   end %if(k==11)
   if((delta-ro)<0)
      ro=ro-2^-k;
      total_cost=total_cost-1;
   end %if((delta-ro)<0)
  end %while ((delta-ro)<10^-k)
end %while((delta-ro)>=10^-k)
x2_rev=xb2-x2;
x2=x2_rev;
% The vector head and vector base has reversed and base carried on to xb2
fprintf('\n\nRotating DL-2(%f,%f) \nwith angle rad_aod2(%f)',x2,y2,delta);
fprintf('\nCalculated arrow head position(%f,%f)',x6,y6);
fprintf('\nCost: %10d',total_cost);




%-------------------------------------------------------------------------
% Stepper vector V21 lay on x-axis and then rotated with angle rad-aod2
% Stepper vector V21 will be used for decrease on R4 vector on vector break
% process of the vector DL-2
```

```
% x21,y21 are the coordinates of the vector head of the stepper vector V21


x21=step_value;

y21=0.0000;

ro=0.0000;


%delta is same as rad_aod2


k=k_rotate;

v=2^-k;

u=2^-(2*k+1);

while((delta-ro)>=(10^-k))

                              total_cost=total_cost+1;

 oldx21=x21;

 x21=x21-x21*u-y21*v;

 y21=y21-y21*u+oldx21*v;

 ro=ro+2^-k;

  while (abs(delta-ro)<(10^-k))

   if (k<11)

     k=k+1;

                               total_cost=total_cost+1;

   end %if (k<11)

   if(k==11)

      break;

   end %if(k==11)
```

```
   if((delta-ro)<0)
      ro=ro-2^-k;
      total_cost=total_cost-1;
   end %if((delta-ro)<0)
  end %while ((delta-ro)<10^-k)
end %while((delta-ro)>=10^-k)

c_x21=step_value*cosd(aod2);
c_y21=step_value*sind(aod2);
fprintf('\n\nRotating V21(%f, %f)',x21,y21);
fprintf('\nCalculated V21=(%f ,%f)',c_x21,c_y21);
fprintf('\nwith angle rad_aod2(%f)',delta);
fprintf('\nCost=%d',total_cost);

%-------------------------------------------------------------------------
% Stepper vector V22 lay on x-axis and then rotated with angle rad-aoa2
% Stepper vector V22 will be used for increase on R3 vector on vector break
% process of the vector DL-2
% x22,y22 are the coordinates of the vector head of the stepper vector V22
x22=step_value;
y22=0.0000;
ro=0.0000;

rad_aoa2=(aoa2*pi)/180;
if (rad_aoa2>pi/2)
```

```
    sign=-1;
    delta=pi-rad_aoa2;
else
    sign=1;
    delta=rad_aoa2;
end %if (rad_aoa2>pi/2)

k=k_rotate;
v=2^-k;
u=2^-(2*k+1);
while((delta-ro)>=(10^-k))
                        total_cost=total_cost+1;
 oldx22=x22;
 x22=x22-x22*u-y22*v;
 y22=y22-y22*u+oldx22*v;
 ro=ro+2^-k;
 while (abs(delta-ro)<(10^-k))
  if (k<11)
     k=k+1;
                                total_cost=total_cost+1;
   end %if (k<11)
   if(k==11)
      break;
   end %if(k==11)
   if((delta-ro)<0)
```

```
      ro=ro-2^-k;
      total_cost=total_cost-1;
   end  %if((delta-ro)<0)
  end %while ((delta-ro)<10^-k)
end %while((delta-ro)>=10^-k)

c_x22=step_value*cosd(aoa2);
c_y22=step_value*sind(aoa2);
fprintf('\n\nRotating V22(%f, %f)',x22,y22);
fprintf('\nCalculated V22=(%f ,%f)',c_x22,c_y22);
fprintf('\nwith angle rad_aoa2(%f)',delta);
fprintf('\nCost=%d',total_cost);
%-----------------------------------------------------------------------




%**************************************************************************
*******
% Vectors DL-1 and Dl-2 will be broken until y1 == y2
%**************************************************************************
*******
 cx1=x1;     % Head-x of R2
 cy1=y1;     % Head-y of R2
 cx2=x2;     % Head-x of R3
```

```
 cy2=y2;     % Head-y of R3
 xtemp=0;    % length-x of R2
 ytemp=0;    % length-y of R2
 x2temp=0;   % length-x of R3
 y2temp=0;   % length-y of R3
fprintf('\n\nCompare T1(%f,%f) & T2(%f,%f)',x1,y1,x2,y2);

if (y1>y2)
fprintf('\nT1(y)>T2(y), so Break DL-1 until T(y1)==T(y2)');
fprintf('\nError=%f \n',err);

 k=k_start;
 while ((cy1-y2)>(10^-k))

                        total_cost=total_cost+1;
    x1=x1-x11;
    y1=y1-y11;                  % reduce stepper vector V11 from R1
    h_xy1= sqrt(x1^2+y1^2);         % current R1 = sqrt(x1^2+y1^2);
                        total_cost=total_cost+113;
    p_xtemp=xtemp;
    p_ytemp=ytemp;
    p_total_cost=total_cost;
    while((Dl1-(h_xy1 +sqrt(xtemp^2+ytemp^2)))>(10^-k))
                        % add stepper vector V12 ro R2
                        % until (R1+R2) == DL1
```

```
                        % current R2 = sqrt(xtemp^2+ytemp^2);
                        total_cost=total_cost+113;
    xtemp=xtemp+x12;
    ytemp=ytemp+y12;
    cx1=x1+xtemp;
    cy1=y1-ytemp;
                        total_cost=total_cost+4;
  end % while((Dl1-(h_xy1 + ...
  while (abs(cy1-y2)<(10^-k))
   if (k<11)
      k=k+1;
      x1=x1+x11;
      y1=y1+y11;              % reduce stepper vector V11 from R1
      xtemp=p_xtemp;
      ytemp=p_ytemp;
      total_cost=p_total_cost;
      cy1=2*y2;
      fprintf('\n fark=%12.11f', (cy1-y2), (10^-k));
                        total_cost=total_cost+1;
   end %if (k<11)
   if(k==11)
     break;
   end %if(k==11)
  end %while ((cy1-y2)<10^-k)
end %while((cy1-y2)>err)
```

```
fprintf('\nNow, DL1(%f, %f)== DL2(%f, %f): %10d',cx1,cy1,cx2,cy2);
fprintf('\nCost=%f\n',total_cost);

else %y2>y1

k=k_start;
fprintf('\nT2(y)>T1(y), so Break DL-2 until T(y2)==T(y1)');
fprintf('\nUsed Error=%12.11f \n',(10^-k));
 while((cy2-cy1)>(10^-k))
   x2=x2+x21;
   y2=y2-y21;                  % reduce stepper vector V21 from R4
                               total_cost=total_cost+5;
   h_xy2= sqrt((x2-xb2)^2+y2^2); % current R4= sqrt((x2-xb2)^2+y2^2);
                               total_cost=total_cost+112;

   fprintf('\n>>k=%d, T2(%6.5f, %6.5f)==>T1(%6.5f, %6.5f); dif=%12.11f;
err=%12.11f',k, x2, y2, x1, y1, (cy2-cy1), (10^-k));
   p_x2temp = x2temp;
   p_y2temp = y2temp;
   p_total_cost = total_cost;
   while((Dl2-(h_xy2 +sqrt(x2temp^2+y2temp^2)))>(10^-k))
                               % add stepper vector V22 to R3
                               % until (R3+R4) == DL2
                               % current R3 = sqrt(x2temp^2+y2temp^2);
                               total_cost=total_cost+113;
```

```
 x2temp=x2temp+x22;
 y2temp=y2temp+y22;
 cx2=x2-x2temp;
 cy2=y2-y2temp;
                        total_cost=total_cost+4;
end % while((Dl2-(h_xy2 + ...
while (abs(cy2-cy1)<(10^-k))
    fprintf('\n==> fark=%12.11f', (cy2-cy1) );
    if (k<11)
      k=k+1;
                            total_cost=total_cost+1;
  end %if (k<11)
  if(k==11)
      break;
  end %if(k==11)
  if((cy2-cy1)<0)
      x2=x2-x21;
      y2=y2+y21;
      x2temp=p_x2temp;
      y2temp=p_y2temp;
      total_cost=p_total_cost;
      cy2=y2+y2temp;
      fprintf('\n fark=%12.11f; error=%12.11f', (cy2-cy1), (10^-k));
  end %if((cy2-cy1)<0)
 end %while ((cy2-cy1)<10^-k)
```

```
  end %while((cy2-cy1)>(10^-k))
fprintf('\n\nNow, DL1(%f, %f)== DL2(%f, %f): %10d',cx1,cy1,cx2,cy2);
fprintf('  dif=%12.11f; err=%12.11f', (cy2-cy1), (10^-k));
fprintf('\nCost=%f\n',total_cost);

%return;
end % else of if(y1>y2)

%-----------------------------------------------------------------
%
% Compare AOA angles and select the greatest to put it into the
% inner loop - greatest angles yield smaller x-axis moves.
%
%-----------------------------------------------------------------
fprintf('\n\nCompare aoa1(%f) & aoa2(%f)',aoa1,aoa2);
%-----------------------------------------------------------------

if (aoa1>aoa2)
   fprintf('\naoa1 > aoa2, First R3 will be broken');
   fprintf('\nThen R4 will be carried to near R3');
   fprintf('\nA(%f, %f), P1(%f,%f), C(%f, %f),
P2(%f,%f)',x1,y1,cx1,cy1,x2,y2,cx2,cy2);

          k=k_start;
  while ((cx2-cx1)>(10^-k))
```

```
    if ((x2<0)||(y2<0))
        break;
    end %if ((x2<0)||(y2<0))
%---- Break the DL2, one step ----------------------------------------
  x2=x2+x21;
  y2=y2-y21;          % reduce stepper vector V21 from R4
                    total_cost=total_cost+3;
  h_xy3 = sqrt((x2-xb2)^2+y2^2);
                    % current R4= sqrt((x2-xb2)^2+y2^2);
                    total_cost=total_cost+114;
  while((Dl2-(h_xy3 +sqrt(x2temp^2+y2temp^2)))>(10^-k))
                    % add stepper vector V22 to R3
                    % until (R3+R4) == DL2
                    % current R3 = sqrt(x2temp^2+y2temp^2);
                    total_cost=total_cost+113;
    x2temp=x2temp+x22;
    y2temp=y2temp+y22;
    cx2=x2-x2temp;
    cy2=y2-y2temp;
                              total_cost=total_cost+4;
  end %while((Dl2-(h_xy3 + ...
%---- Breake DL1 several steps until cy1==cy2 -------------------------
  while((cy1-cy2)>(10^-k))
   if (cnt2==test_step)
    fprintf('\n\t\t\t\t\t C(%f, %f), P2(%f,%f)\n',x2,y2,cx2,cy2);
```

```
    fprintf('\n  xtemp/ytemp(%f, %f) - x2t/y2t(%f, %f)', xtemp, ytemp, x2temp,
y2temp);
    fprintf('\n  DL1-G(%f), DL1-C(%f)',Dl1,
sqrt(x1^2+y1^2)+sqrt(xtemp^2+ytemp^2));
    fprintf('\n  DL2-G(%f), DL2-C(%f)',Dl2,sqrt((x2-
xb2)^2+y2^2)+sqrt(x2temp^2+y2temp^2));
   end %if (cnt2==test_step)
     x1=x1-x11;
     y1=y1-y11;          % reduce stepper vector V11 from R1
                    total_cost=total_cost+2;
     h_xy4= sqrt(x1^2+y1^2);
                    % current R1= sqrt(x1^2+y1^2);
                    total_cost=total_cost+113;
    if(x1<0)
       fprintf('\n Error x1(%f)<0',x1);
       break; return;
    end %if(x1<0)

    if((Dl1-(h_xy4 +sqrt(xtemp^2+ytemp^2)))<(10^-k))
       fprintf('\n Errordan küçük ---> line 475');
       continue;
    end %if((Dl1-(h_xy4 +
    while((Dl1-(h_xy4 +sqrt(xtemp^2+ytemp^2)))>(10^-k))
                    % add stepper vector V12 to R2
                    % until (R1+R2) == DL1
```

```
                    % current R2 = sqrt(xtemp^2+ytemp^2);
                                total_cost=total_cost+115;
      xtemp=xtemp+x12;
      ytemp=ytemp+y12;
      cx1=x1+xtemp;
      cy1=y1-ytemp;
                                total_cost=total_cost+4;



      if(abs(cx1-cx2)>(10^-k))
        fprintf('\n(Err)P1(%f,%f) & P2(%f,%f)',cx1,cy1,cx2,cy2);


      end %if((cx1-cx2)>(10^-k))
    end %while((Dl1-(h_xy4 + ...
      if(cnt2==test_step)
        fprintf('\n...R1=%f,',h_xy5);
        fprintf('\nA(%f, %f), P1(%f,%f)\n',x1,y1,cx1,cy1);
        fprintf('\n  xtemp/ytemp(%f, %f) - x2t/y2t(%f, %f)', xtemp, ytemp, x2temp,
y2temp);
        fprintf('\n  DL1-G(%f), DL1-C(%f)',Dl1,
sqrt(x1^2+y1^2)+sqrt(xtemp^2+ytemp^2));
        fprintf('\n  DL2-G(%f), DL2-C(%f)',Dl2,sqrt((x2-
xb2)^2+y2^2)+sqrt(x2temp^2+y2temp^2));
        %pause;
      end %if(cnt2==test_step)
```

```
   end %while((cy1-cy2)>(10^-k))


         while ((cx2-cx1)<(10^-k))
     if (k<11)
        k=k+1;
                              total_cost=total_cost+1;
     end %if (k<11)
     if(k==11)
        break;
     end %if(k==11)
   end % while ((cx2-cx1)<(10^-k))
 end %while((cx2-cx1)>(10^-k))


 fprintf('\n\nMS(%f,%f),MS(%f,%f) Found!(aoa1<aoa2)',cx1,cy1,cx2,cy2);
 fprintf('\nMS(%f,%f) Given',xm,ym);
 fprintf('\nCost=%10d',total_cost);


else %aoa2>aoa1
%-------------------------------------------------------------
%
%-- Break the DL1, one step ----------------------------------
%
%-------------------------------------------------------------
fprintf('\naoa2 > aoa1, Smaller angle gives larger x, smaller y moves');
fprintf('\nSo, First R1 will be broken');
```

```
fprintf('\nThen R2 will be carried to near R1');
fprintf('\nP1(%f,%f), P2(%f,%f)\n',cx1,cy1,cx2,cy2)



k=k_start;
while ((cx2-cx1)>(10^-k))
if(cnt2==test_step)
    fprintf('\nP1(%f, %f) - P2(%f,%f)',cx1,cy1,cx2,cy2);
    fprintf('\n  m_err=%f m..',m_err);
    fprintf('\n  xt/yt(%f, %f) - xt2/yt2(%f, %f)', xtemp, ytemp, x2temp, y2temp);
    fprintf('\n  DL1-G(%f), DL1-C(%f)',Dl1,
sqrt(x1^2+y1^2)+sqrt(xtemp^2+ytemp^2));
    fprintf('\n  DL2-G(%f), DL2-C(%f)\n',Dl2,sqrt((x2-
xb2)^2+y2^2)+sqrt(x2temp^2+y2temp^2));
    h_xy5 = sqrt(x1^2+y1^2);
    fprintf('\n h_xy5=%f,',h_xy5);
    fprintf('\n Dl1-(h_xy5+R3)=%f,',(Dl1-(h_xy5 +sqrt(xtemp^2+ytemp^2))));
    pause;
end; %if(cnt2==test_step)
% reduce stepper vector V11 from R1
    x1=x1-x11;
    y1=y1-y11;
                        total_cost=total_cost+2;
    h_xy5 = sqrt(x1^2+y1^2);
                        % current R1= sqrt(x1^2+y1^2);
```

```
                    total_cost=total_cost+113;

 if(cnt2==test_step)
    fprintf('\n...x1=%f, y1=%f, h_xy5=%f,',x1,y1,h_xy5);
 end %if(cnt2==test_step)

 if((Dl1-(h_xy5 +sqrt(xtemp^2+ytemp^2)))<(10^-k))
    fprintf('\n --> error on line 673');
    break; %return;
 end %if((Dl1-(h_xy5 + ...

while((Dl1-(h_xy5 +sqrt(xtemp^2+ytemp^2)))>(10^-k))
                    % add stepper vector V12 to R2
                    % until (R1+R2) == DL1
                    % current R2 = sqrt(xtemp^2+ytemp^2);
                    total_cost=total_cost+113;
   xtemp=xtemp+x12;
   ytemp=ytemp+y12;
   cx1=x1+xtemp;
   cy1=y1-ytemp;
                        total_cost=total_cost+4;
 if(cnt2==test_step)
    fprintf('\n...x1=%f, y1=%f, h_xy5=%f,',x1,y1,h_xy5);
 end; %if(cnt2==test_step)
end; %while((Dl1-(h_xy5 + ...
```

```
%---- Breake DL2 several steps until cy1==cy2 -------------------------
   while((cy2-cy1)>(10^-k))
   % reduce stepper vector V21 from R4
   x2=x2+x21;
   y2=y2-y21;
                                total_cost=total_cost+2;
   h_xy6 = sqrt((x2-xb2)^2+y2^2);
   % current R4= sqrt((x2-xb2)^2+y2^2);
                           total_cost=total_cost+114;

   if ((Dl2-( h_xy6 +sqrt(x2temp^2+y2temp^2)))<(10^-k))
       continue;
   end; %if ((Dl2-( h_xy6 + ...

   while((Dl2-( h_xy6 +sqrt(x2temp^2+y2temp^2)))>(10^-k))
                            % add stepper vector V22 to R3
                            % until (R3+R4) == DL2
                            % current R3 = sqrt(x2temp^2+y2temp^2);
                               total_cost=total_cost+113;

    x2temp=x2temp+x22;
    y2temp=y2temp+y22;
    cx2=x2-x2temp; %xb2 hatası olduğunu düşünürüm.
    cy2=y2-y2temp;
                           total_cost=total_cost+4;
```

```
    %fprintf('+ %f +',cx2);
   end %while((Dl2-( h_xy6 + ...
  end %while((cy2-cy1)>(10^-k))
  while (abs(cx2-cx1)<(10^-k))
    if (k<11)
       k=k+1;
                          total_cost=total_cost+1;
    end %if (k<11)
    if(k==11)
       break;
    end %if(k==11)
   end % while ((cx2-cx1)<10^-k)
 end %while((cx2-cx1)>(10^-k))

fprintf('\n\nMS(%f,%f),MS(%f,%f) Found!(aoa1<aoa2)',cx1,cy1,cx2,cy2);
fprintf('\nMS(%f,%f) Given',xm,ym);
fprintf('\nCost=%10d',total_cost);
end % if(aoa1>aoa2)

% ------------------------------------------------------------------
% Z-Axis Processes
% ------------------------------------------------------------------
% Step 1: Vector angle rotation, finding the (x,z) coordinates with
%         angle AOD3, AOA3 and length DL3
% initial inputs
```

```
%step_value=step_value*0.1000;
rad_aod3=(aod3*pi)/180;
ro=0.0000;
x8=Dl3;
z8=0.0000;
if (rad_aod3>pi/2)
    sign=-1;
    delta=pi-rad_aod3;
else
    sign=1;
    delta=rad_aod3;
end %if (rad_aod3>pi/2)

k=k_rotate;
v=2^-k;
u=2^-(2*k+1);
while((delta-ro)>10^-k)
   oldx8=x8;
   x8=x8-x8*u-z8*v;
   z8=z8-z8*u+oldx8*v;
   ro=ro+2^-k;
                              total_cost=total_cost+1;
          while (abs(delta-ro)<10^-k)
      if (k<11)
        k=k+1;
```

```
                    total_cost=total_cost+1;
      end %if (k<11)
      if(k==11)
       break;
      end %if(k==11)
      if((delta-ro)<0)
       ro=ro-2^-k;
       total_cost=total_cost-1;
      end  %if((delta-ro)<0)
     end % while ((delta-ro)<10^-k)
end %while((delta-ro)>10^-k)
fprintf('\n\nRotating DL-3(%f,%f)',x8,z8);
fprintf('\nwith angle rad_aod3(%f)',delta);
fprintf('\nCalculated arrow head position(%f,%f)',x7,z7);
fprintf('\nCost: %10d',total_cost);

%------------------------------------------------------------------------
% Stepper vector V81 lay on x-axis and then rotated with angle rad-aod3
% Stepper vector V81 will be used for decrease on R5 vector on vector break
% process of the vector DL-1
% x81,y81 are the coordinates of the vector head of the stepper vector V81

x81=step_value;
z81=0.0000;
ro=0.0000;
```

```
rad_aod3=(aod3*pi)/180;
if (rad_aod3>pi/2)
    sign=-1;
    delta=pi-rad_aod3;
else
    sign=1;
    delta=rad_aod3;
end %if (rad_aod3>pi/2)

k=k_rotate;
v=2^-k;
u=2^-(2*k+1);
while((delta-ro)>=(10^-k))
  oldx81=x81;
  x81=x81-x81*u-z81*v;
  z81=z81-z81*u+oldx81*v;
  ro=ro+2^-k;
                                                    total_cost=total_cost+1;
  while (abs(delta-ro)<(10^-k))
    if (k<11)
     k=k+1;
                          total_cost=total_cost+1;
    end %if (k<11)
    if(k==11)
      break;
```

```
    end %if(k==11)     end %while ((delta-ro)<10^-k)
    if((delta-ro)<0)
      ro=ro-2^-k;
      total_cost=total_cost-1;
    end %if((delta-ro)<0)
  end %while ((delta-ro)<(10^-k))
end %while((delta-ro)>=(10^-k))
c_x81=step_value*cosd(aod3);
c_z81=step_value*sind(aod3);
fprintf('\n\nRotating V81(%f, %f)',x81,z81);
fprintf('\nCalculated V81=(%f, %f)',c_x81,c_z81);
fprintf('\nwith angle rad_aod3(%f)',delta);
fprintf('\nCost=%d',total_cost);

%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
% Stepper vector V82 lay on x-axis and then rotated with angle rad-aoa3
% Stepper vector V82 will be used for increase on R6 vector on vector break
% process of the vector DL-3
% x82,y82 are the coordinates of the vector head of the stepper vector V82

x82=step_value;
z82=0.0000;
ro=0.0000;
```

```
rad_aoa3=(aoa3*pi)/180;
if (rad_aoa3>pi/2)
   sign=-1;
   delta=pi-rad_aoa3;
else
   sign=1;
   delta=rad_aoa3;
end  %if (rad_aoa3>pi/2)

k=k_rotate;
v=2^-k;
u=2^-(2*k+1);
while((delta-ro)>(10^-k))
  oldx82=x82;
  x82=x82-x82*u-z82*v;
  z82=z82-z82*u+oldx82*v;
  ro=ro+2^-k;
                          total_cost=total_cost+1;
  while (abs(delta-ro)<(10^-k))
    if (k<11)
     k=k+1;
                              total_cost=total_cost+1;
    end %if (k<11)
    if(k==11)
      break;
```

```
    end %if(k==11)
    if((delta-ro)<0)
       ro=ro-2^-k;
       total_cost=total_cost-1;
    end %if((delta-ro)<0)
  end % while ((delta-ro)<10^-k)

end %while((delta-ro)>=(10^-k))

c_x82=step_value*cosd(aoa3);
c_z82=step_value*sind(aoa3);
fprintf('\n\nRotating V82(%f, %f)',x82,z82);
fprintf('\nCalculated V82=(%f, %f)',c_x82,c_z82);
fprintf('\nwith angle rad_aoa3(%f)',delta);
fprintf('\nCost=%d',total_cost);

%-----------------------------------------------------------------------
%***********************************************************************
*******
% Vectors DL-3 will be broken until cx1 == cx8
%***********************************************************************
*******
fprintf('\n\nVector DL-3 will be broken untill cx8==cx1');
 cx8=x8;  % Head-x of R6
 cz8=z8;  % Head-z of R6
```

```
 xtemp=0; % length-x of R6
 ztemp=0; % length-z of R6

fprintf('\nCompare P1(%f, %f) & P3(%f, %f)',cx1,cy1,cx8,cz8);
 if(cx1>cx8)
  fprintf('\nP1x(%f) > P3x(%f)',cx1,cx8);
  cnt3=0;

  k=k_start;
  while ((cx1-cx8)>(10^-k))
     % reduce stepper vector V81 from R5
     x8=x8-x81;
     z8=z8-z81;
                                  total_cost=total_cost+3;
     h_xy7 = sqrt(x8^2+z8^2);
           % current R5 = sqrt(x8^2+z8^2);
                                  total_cost=total_cost+113;
    while((Dl3-(h_xy7 +sqrt(xtemp^2+ztemp^2)))>(10^-k))
                                       % add stepper vector V82 ro R6
                                  % until (R5+R6) == DL3
                                  % current R6 = sqrt(xtemp^2+ztemp^2);
                                   total_cost=total_cost+113;
       xtemp=xtemp+x82;
       ztemp=ztemp+z82;
       cx8=x8+xtemp;
```

```
      cz8=z8-ztemp;
                                total_cost=total_cost+4;
   end % while((Dl3-(h_xy7 +sqrt(xtemp^2+ztemp^2)))>10^-k)
          while (abs(cx1-cx8)<(10^-k))
     if (k<11)
        k=k+1;
                                total_cost=total_cost+1;
     end %if (k<11)
     if(k==11)
        break;
     end %if(k==11)
   end % while ((cx1-cx8)<(10^-k))

     if((cx8<0) || (cz8<0))
         break;
     end %if((cx8<0) || (cz8<0))
  end %while((cx1-cx8)>(10^-k))

%    fprintf('\n(*)Cost> Found z in (cx1>cx8) sitution: %10d',total_cost);
fprintf('\n\nMS(x,y,z) FOUND for SAMPLE-%d (cx8<cx1)',cnt);
fprintf('\nMS-1(%f, %f, %f)',cx1,cy1,cz8);
fprintf('\nMS-2(%f, %f, %f)',cx2,cy2,cz8);
fprintf('\nMS(%f, %f, %f) Given',xm,ym,zm);
fprintf('\nCost=%10d',total_cost);
```

```
else %  else of if(cx1>cx8)
    fprintf('\n P3x(%f) > P1x(%f)',cx8,cx1);
    %fprintf('\n(***)(cx1<=cx8) sitution');

          k=k_start;
    while ((cx8-cx1)>(10^-k))
    x8=x8-x81;
    z8=z8-z81;
                              total_cost=total_cost+3;
    h_xy8 = sqrt(x8^2+z8^2);
                              total_cost=total_cost+111;
    while((Dl3-(h_xy8 +sqrt(xtemp^2+ztemp^2)))>(10^-k))
                              total_cost=total_cost+113;
     xtemp=xtemp+x82;
     ztemp=ztemp+z82;
     cx8=x8+xtemp;
     cz8=z8-ztemp;
                              total_cost=total_cost+4;
   end % while((Dl3-(h_xy8 ...

         while (abs(cx1-cx8)<(10^-k))
    if (k<11)
       k=k+1;
                              total_cost=total_cost+1;
    end %if (k<11)
```

```
      if(k==11)
          break;
      end %if(k==11)
    end % while ((cx8-cx1)<10^-k)

    if((cx8<0) || (cz8<0))
          break;
    end %if((cx8<0) || (cy8<0))
  end % while(cx8-cx1)>(10^-k))

%fprintf('\n(*)Cost> Finding z in (cx1<cx8) sitution: %10d',total_cost);
fprintf('\n\nMS(x,y,z) FOUND for SAMPLE-%d (cx1<cx8)',cnt);
fprintf('\nMS-1(%f, %f, %f)',cx1,cy1,cz8);
fprintf('\nMS-2(%f, %f, %f)',cx2,cy2,cz8);
fprintf('\nMS(%f, %f, %f) Given',xm,ym,zm);
fprintf('\nCost=%10d',total_cost);

end %if(cx1>cx8)
if (cy1>cy2)
    cy=cy2;
else
    cy=cy1;
end %if (cy1>cy2)

cy=(cy1+cy2)/2;
```

```
%cy=cy1;
%SON ADIMLAR SONU
t_err_x=t_err_x+ abs(xm-cx1);
t_err_y=t_err_y+ abs(ym-cy);
t_err_z=t_err_z+abs(zm-cz8);
t_cost=t_cost+total_cost;
fprintf(fw,'%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t
%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8.5f\t%8d\n',xm,ym,zm,cx1,cy1,cx2,cy2,cx8,
cz8,x1,y1,x2,y2,(xm-cx1),(ym-cy),(zm-cz8), total_cost );
end  %while ~feof(fr)

%end %for
fprintf(fw,'\n k=%d, m_err=%f, ste_V=%f',k, m_err, step_value);
fprintf(fw,'\n Average Errors: mx-cx: %f, my-cy=%f, mz-cz:%f, Avg-Cost: %f
\n',t_err_x/cnt, t_err_y/cnt, t_err_z/cnt, t_cost/cnt);
fprintf('\n(*)Operation Completed\n\n\n');
fclose(fr);
fclose(fw);
load gong
sound(y,Fs)
%hold off
h = msgbox('Operation Completed');

%msgbox(strcat('x11:',num2str(x11),' y11:',num2str(y11),' x12:',num2str(x12),'
y12:',num2str(y12)));
```

```
%plot(x1,y1,'Color', [0.0, 1.0, 0.0], 'Marker', '+ ');
```

Appendix D: List of Publications

- C. Yağlı, E. Özen, A. Akkeleş, Fast hardware-oriented algorithm for 3D positioning in line-of-sight and single bounced non-line-of-sight environments, Tübitak, Turkish Journal of Electrical Engineering & Computer Sciences, March 12, 2021., DOI: 10.3906/elk-2008-98

- C. Yağlı, E. Özen, A. Akkeleş, The use of CORDIC variants in self-location estimation process via line of sight and single bounced non line of sight signal arriving approaches, in Proceedings of the International Asian Congress on Contemporary Sciences-V, Nakhchivan, Azerbaijan; (2021 June)