ALGORITHM DEVELOPMENT FOR SENSOR DATA PROCESSING AND TRANSMITTER POSITION ESTIMATION

by

ALEXIOS STARIDAS

B.A., University of Crete, 2019

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SCHOOL OF ENGINEERING

HELLENIC MEDITERRANEAN UNIVERSITY

2023

Approved by:

Major Professor
ATHANASIOS MALAMOS

Copyright

Abstract

In the last years, localization has evolved into a topic of increasing scientific interest, since the urge to be able to detect a position at any time has become a matter of prime importance. Many applications nowadays use localization, including GPS positioning, mobile phone technologies and robotics. One of the main methods of localization – and the topic of this thesis – is triangulation. Triangulation is used extensively since it offers an efficient way to determine someone's position just by taking bearings on them. The current thesis will be conducted on the area of transmitters' range free localization using triangulation. In a wide area of interest (AOI), where many transmitters may operate, it is important for a Fixed Sensors Network (FSN) to be able to estimate the transmitters' positions. If more than one transmitter operate in the AOI, then at least three sensors are needed to cover that area. In the case of obstacles/interference in the AOI, more than three sensors are needed for adequate coverage of the area. The sensors are organized over an area of several square kilometers, collecting data in real time. Algorithms need to be developed for the analysis and processing of the collected data, as well as the transmitter's position estimation and projection on the map. The collected data needs to be analyzed, in order to find all high-intensity bearings corresponding to the transmitter. The data must then be used in triangulation algorithms, in order to find all triangulation areas of the transmitter's potential position. Finally, using the triangulation areas, the transmitter's position must be estimated and its geographical position must be projected on the map in real time. Following what has been described, the thesis' structure is formed by the following stages: a) sensor data analysis/processing, b) triangulation areas' determination and c) transmitter's position estimation and its projection on the map.

Σύνοψη

Τα τελευταία χρόνια, ο εντοπισμός θέσης έχει αναχθεί σε θέμα αυξανόμενου επιστημονικού ενδιαφέροντος, καθώς η ανάγκη για την ανίχνευση θέσης σε οποιαδήποτε στιγμή έχει καταστεί θέμα πρωταρχικής σημασίας. Στις μέρες μας πολλές εφαρμογές χρησιμοποιούν εντοπισμό, όπως το GPS, η κινητή τηλεφωνία και η ρομποτική. Μία από τις κύριες μεθόδους εντοπισμού – και το κύριο θέμα της παρούσας εργασίας – είναι η μέθοδος του τριγωνισμού. Η μέθοδος του τριγωνισμού χρησιμοποιείται εκτεταμένα, καθώς προσφέρει έναν αποτελεσματικό τρόπο για να προσδιορίσουμε την θέση κάποιου, απλά εφαρμόζοντας διοπτεύσεις προς το μέρος τους. Η παρούσα εργασία θα διεξαχθεί στον τομέα του προσδιορισμού θέσης πομπών, με την χρήση μεθόδων τριγωνισμού. Σε μια ευρεία περιοχή ενδιαφέροντος, όπου μπορεί να λειτουργούν πολλοί πομποί, είναι σημαντικό για ένα Δίκτυο Σταθερών Αισθητήρων (Fixed Sensor Network – FSN) να είναι σε θέση να εκτιμήσει/εντοπίσει την θέση των πομπών. Εάν περισσότεροι από ένας πομποί υπάρχουν στην περιοχή ενδιαφέροντος, τότε χρειάζονται τουλάχιστον τρεις (3) αισθητήρες για την επαρκή κάλυψη της περιοχής. Σε περίπτωση εμποδίων/παρεμβολών στην περιοχή, τότε περισσότεροι από τρεις (3) αισθητήρες χρειάζονται για την επαρκή κάλυψη της περιοχής. Οι αισθητήρες είναι οργανωμένοι σε μια περιοχή αρκετών τετραγωνικών χιλιομέτρων, συλλέγοντας δεδομένα σε πραγματικό χρόνο. Για την ανάλυση και την επεξεργασία των δεδομένων, καθώς και για τον προσδιορισμό και την προβολή της θέσης ενός πομπού στον χάρτη, καθίσταται απαραίτητη η ανάπτυξη κατάλληλων αλγορίθμων. Τα δεδομένα που συλλέγονται πρέπει να αναλυθούν, ώστε να εντοπιστούν όλες οι διοπτεύσεις μεγάλης έντασης πεδίου που αντιστοιχούν στον εκάστοτε πομπό. Έπειτα, τα δεδομένα πρέπει να εισαχθούν σε αλγόριθμους τριγωνισμού, έτσι ώστε να εντοπιστούν όλες οι δυνατές περιοχές τριγωνισμού, μέσα στις οποίες ενδέχεται να υπάρχει ο πομπός. Τέλος, χρησιμοποιώντας τις περιοχές τριγωνισμού, πρέπει να εντοπιστεί η γεωγραφική θέση του πομπού και να προβληθεί το γεωγραφικό του στίγμα στον χάρτη σε πραγματικό χρόνο. Ακολουθώντας, λοιπόν, τα όσα διατυπώθηκαν έως τώρα, η δομή της εργασίας διαμορφώνεται στα ακόλουθα στάδια: α) ανάλυση/επεξεργασία δεδομένων των αισθητήρων, β) προσδιορισμός περιοχών τριγωνισμού και γ) προσδιορισμός/εκτίμηση του γεωγραφικού στίγματος του πομπού και προβολή του στον χάρτη.

Table of Contents

List of Figures

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Emmanouil Antonidakis, for trusting me from the very beginning of our cooperation and friendship, back in our first meeting in 2018. His continuous support, advice and guidance during all these years have helped me a lot and for that, I am always grateful.

Chapter 1 - Introduction

1.1 Chapter Overview

The introductory chapter provides an overview of the topics that will be addressed in the current thesis. The thesis' focus and contribution to existing knowledge will be presented. Moreover, a brief summary of the chapters' contents will be provided, in order to prepare the reader for what to expect from the thesis.

1.2 Focus

The main focus of the thesis will be sensor data analysis and processing, in order to perform localization using the method of triangulation. There will be a Fixed Sensor Network (FSN) provided, with its sensors collecting amounts of data over a large area of several square kilometers on the surface of the Earth (area of interest – AOI) in real time. The sensors will be connected wirelessly via 4G, thus declaring our network a Wireless Sensor Network (WSN).

Our aim is to process and analyze the collected data and use it in triangulation algorithms, in order to estimate a transmitter's position that may potentially enter our AOI. Moreover, upon estimating the transmitter's position, the next step is to project its geographical position on a map in real time, in order to able to keep track of the transmitter's activity while being in our AOI.

1.3 Contribution to knowledge

The current thesis aims to offer the following contributions:

i. An algorithm for analyzing and processing the data collected by various sensors in a Wireless Sensor Network (WSN), in order to find all high-intensity bearings for transmitters entering the area of interest (AOI)

ii. An algorithm for finding all triangulation areas, in case a transmitter/number of transmitters enter the area of interest (AOI), using triangulation methods and the high-intensity bearings, as defined by the previous algorithm

iii. An algorithm for estimating the transmitters' locations and projecting their geographical locations on the map

iv. A software combining all three (3) algorithms into one (1) program, divided into three (3) steps: a) data analysis, b) triangulation areas' determination and c) position estimation

1.4 Thesis organization

The current thesis is organized in five (5) chapters:

i. ***Chapter 1 – Introduction:*** contains the introductory information of the thesis

ii. ***Chapter 2 – Related Work:*** presents the related work that has been conducted, concerning the topics of the thesis, such as Wireless Sensor Networks (WSNs) and the problem of localization

iii. ***Chapter 3 – Methodology and Technologies:*** presents the localization method that will be used in the current thesis (triangulation) and the technologies that will be used in the development phase

iv. ***Chapter 4 – Project Analysis:*** presents the problem statement, the algorithms' description and the experiment/results of the thesis

v. ***Chapter 5 – Conclusion/Future Work:*** contains a summary of what was presented in the thesis and lists the next steps that will be implemented as future work

Chapter 2 - Background / Related Work

2.1 Chapter Overview

The current chapter gives a detailed overview of the related work that has been conducted on the fields of Wireless Sensor Networks (WSNs) and localization. In the last years, the problem of localization has evolved into a topic of increasing scientific interest. Moreover, due to the concurrent increase of Sensor Network applications in many fields, the urge to be able to detect a transmitter's position at any given time has been deemed a matter of great importance.

One of the many localization techniques – and the topic of the current thesis – is the process of triangulation. In this chapter, we will present some information on both general topics – such as Sensor Networks – and on the topics regarding this thesis, such as Fixed Sensor Networks (FSNs), Wireless Sensor Networks (WSNs), range-free localization and fuzzy systems that use the process of triangulation in order to perform localization.

2.2 Sensor Networks

In recent years, various types of Sensor Networks have been used in a wide range of fields, including environmental monitoring, medical, scientific, military purposes etc. A Sensor Network can be defined as a group of sensor nodes that cooperate in order to carry out specific tasks [1]. Depending on the use the network is intended for, the architecture, topology, scalability and sensor characteristics may vary. In contrast to conventional networks, Sensor Networks rely on dense deployment and coordination.

There are numerous different types of Sensor Networks, depending on their state, the types of sensors they consist of, their topologies and their scale. Networks deployed in broad areas may face challenges regarding scalability and area coverage, though these issues can be successfully confronted by the correct choice of sensors and the network's tailoring to the demands of its intended use. It is worth noting that each network is designed for a particular purpose, therefore one network may not be suitable for another purpose.

In the current thesis, we will use a Fixed Sensor Network (FSN), in which all sensors communicate wirelessly with each other via 4G. The network's deployment can occur anywhere on the surface of the Earth, where wiring connection may not be feasible. Therefore, wireless communication between the sensors is required, declaring our network also a Wireless Sensor Network (WSN). In Fixed Sensor Networks, sensors are deployed in fixed positions, therefore their geographical positions are already known. The positions have been acquired with GPS, so the network system is aware of their coordinates, meaning that no further action is required, in order to determine the sensors' positions. The network is deployed within a predetermined geographical area, also known as area of interest (AOI), to collect large amounts of data. The data is then forwarded to a database for analysis.

In our network, the sensors cooperate wirelessly with one another, via a Wireless Local Area Network (WLAN). As described in [2], there are three (3) basic components that constitute a WLAN: a) the access point (connecting the client to the Internet), b) the Wireless Medium and the Client/Station. The client can be a laptop, tablet or a smart phone with Wi-Fi. The communication between the sensors is very critical and depends on the topology and the internal architecture of the wireless sensor nodes [3]. Moreover, in order to optimize the total energy consumption of the network, the positioning of the base station that collects and handles the data is also crucial [4].

In the unfortunate scenario of a sensor/number of sensors failure in a WSN, critical information might be lost. The related work on connectivity recovery describes a group of sensors using their movement, in order to contribute to the recovery process. In [5], authors presented a novel distributed algorithm named "Autonomous Repair (AuR)" – regarding a WSN in which sensors are movable – that enables connectivity restoration through local coordination among sensors in the individual segments. The neighbors of a failed node collaboratively decide on the network's recovery, by moving one or more healthy nodes.

Note: The information stated above is not directly connected to the current thesis' focus. It concerns topics that refer to Sensor Network organization and technical details about the network's orchestration. However, it is important to be aware of our network's contents and behavior, since it may offer valuable help for future work and optimizations, both on the network's orchestration/deployment and recovery.

2.3 Fuzzy Logic in Wireless Sensor Networks

Ongoing research that implements Fuzzy Logic theory for detection in WSNs exists in many sectors, including fire detection and warning systems. In [6], a Fire Monitoring and Warning System (FMWS) was presented – based on Fuzzy Logic – for the detection of a real existent and dangerous fire incident, transmitting alerts to the Fire Management System (FMS). Energy consumption is a crucial problem in a WSN, since it has a direct impact on the network's operation and lifetime. In [7], authors presented a novel energy-sufficient method which uses Fuzzy Logic applied on cluster heads (CH) of WSNs, focusing on cluster formation. The presented model proved that the proposed protocol improves network lifetime when compared with the low-energy adaptive clustering hierarchy protocol.

In [8], authors used a Fuzzy Logic algorithm to estimate the sensor nodes' positions in a WSN. Despite the use of a fuzzy controller and a specific defuzzification method, it was noted that there are still numerous fundamental issues that need to be solved for the development of WSNs' technologies. In [9], a Fuzzy Logic Cluster Leach Protocol (FUZZY-LEACH) was applied, that used a Fuzzy Logic Inference System (FIS) in the cluster process. It was proven that the network's energy consumption decreases when multiple parameters in the cluster are used. Fuzzy Logic in WSNs enhances decision-making, contributes to resource consumption and generally increases the network's performance through effective deployment, localization, cluster head selection, security etc. [10]. In [11], Kapitanova et al. demonstrated that Fuzzy Logic in a WSN monitoring a fire event (fire and smoke) can provide a more accurate event detection.

By far, the most fuzzy-based reasoning used in fuzzy-based positioning systems is the Fuzzy Inference (FI). The two (2) most commonly used aggregation functions are the Mamdani-Type and the Sugeno-Type Fuzzy Inference Systems (FIS) [12]. In [13], Garcia-Jimenez et al. introduced a specific generalization of the Mamdani-Type FIS, by using overlap functions and overlap indices. The fuzzy method that was presented – based on overlap indices – aims to improve fire detection by utilizing a WSN and analyzing fire lightness and distance. For the same purpose, two (2) fuzzy techniques based on temporal characteristics were proposed in [14]. Through the use of a WSN and the incorporation of Fuzzy Logic in the network's sensor nodes, evidence of fire is analyzed, this time however by evaluating and comparing both previous and present temperature values.

In [15], authors explored energy consumption of a WSN by using a fuzzy Genetic Algorithm (GA) clustering and Ant Colony Optimization (ACO) routing. Fuzzy Logic was used for cluster formation and clusters' head selection. A Genetic Algorithm was used for the optimum generation of fuzzy rules and the tuning of the output value of the Fuzzy Logic's membership functions, whereas the proposed Ant Colony Optimization (ACO) was used to route the information in the shortest path between the cluster heads and the base station. The results showed an improvement in the energy level of a single node and an enhancement in the overall network's lifetime.

2.4 Localization in Wireless Sensor Networks

There are two (2) main categories of localization in Wireless Sensor Networks: a) Target/Source localization and Node self-localization [1]. Depending on the area of interest (AOI), Target/Source localization can be classified in two (2) categories:

i. **Indoor**: target may be a human or device moving inside a house
ii. **Outdoor**: target may be a vehicle or aircraft

There are also cases of underwater localization, where the target may be a sea animal. Depending on the number of targets, Target/Source localization can be either Single-Target or Multiple-Target. A lot of research has been conducted in the Single-Target localization field. For instance, the source location can be estimated by the Angle of Arrival (AOA) [2] or the Time Difference of Arrival [3], [4], [16]. However, not so much research has been conducted in the field of Multiple-Target localization and most papers are based on the likelihood estimator, thus not providing efficient results.

Localization in WSNs can also be range-based or range-free. In range-based localization, distance between nodes needs to be calculated, whereas in range-free techniques localization is related between nodes and topological information of sensor nodes [17]. Range-based schemes are distance-estimation and angle-estimation-based techniques. Classic methods of range-based localization include Time of Arrival (TOA), Time Difference of Arrival (TDOA), Angle of Arrival (AOA) and Received Signal Strength (RSS) [18]–[22]:

i. **Time of Arrival (TOA)**: measures travel times of signals between anchor nodes and an unknown node
ii. **Time Difference of Arrival (TDOA)**: locates arrival time of signals by measuring the difference between anchor nodes and an unknown node, thus achieving high ranging accuracy
iii. **Angle of Arrival (AOA)**: estimates location of an unknown node by using the angles between anchor nodes and the node
iv. **Received Signal Strength (RSS)**: distance between transmitter and receiver is estimated by measuring signal strength at the receiver

In contrast to range-based localization, range-free localization is the preferred option for WSNs, since it is a more inexpensive method. Range-based approaches require high-cost hardware and consume more energy. Range-free techniques provide a cost-effective alternative, since only a fraction of sensor nodes – anchor nodes – are used in the localization process. Classic range-free algorithms include the Distance Vector (DV) Hop, the Centroid and the Approximate Point in Triangulation (APIT):

i. **Distance Vector (DV) Hop**: range between nodes is estimated using hop count
ii. **Centroid**: uses proximity-based grained localization algorithms to estimate an unknown node's location
iii. **Approximate Point in Triangulation (APIT)**: uses regional determination to estimate an unknown node's location

It has been shown that the Centroid algorithm is very cost-effective, whereas DV Hop is efficient in larger networks [23]–[25].

The current thesis will be conducted on the area of range-free localization, using triangulation and a variation of the Centroid algorithm. Despite some notable research that has been conducted in the area of triangulation in WSNs, there still remains a lot to be discovered. In [26], a novel work on transmitters' localization using triangulation with a FSN was proposed. In order for the network to be able to continue working normally, the detection complexity was tackled by finding the optimal detecting sensor radius. The system was able to keep a high detection rate, while also identifying potential new transmitters that would enter its area of interest (AOI).

As the triangulation problem becomes highly complicated in networks with many sensors and transmitters, an adequate grid topology is needed, in order to tackle detection complexity. In [27], authors proposed a network grid topology for a FSN that uses triangulation, in order to perform localization. Moreover, concepts like sensor blindness and overall network blindness were analyzed. Finally, in [28], Sfendourakis et al. presented a FSN that implements Fuzzy Logic, in order to perform localization with triangulation and increase the network's detection rate. Furthermore, the authors proved that in the case of an increase in the transmitters' number in the AOI, the proposed system was able to maintain its high performance by utilizing additional groups of sensors in a sub-region of the AOI. Therefore, even in the event of the network's saturation by many transmitters in one region, new transmitters can still be detected when entering the AOI. In the next chapter, the basic triangulation principles and the methodology of the current thesis will be further analyzed.

Chapter 3 - Methodology and Technologies

3.1 Chapter Overview

The current chapter gives a detailed description of the methodology, basic concepts and principles that will be used for the development of the thesis. Despite the numerous localization categories that were mentioned in Chapter 2 (section 2.4), we will use triangulation as a localization method, in order to determine a transmitter's position. Moreover, we will define the basic principles of triangulation, along with a brief analysis of the sensors that will be used for the needs of the current thesis. Last but not least, we will refer to the technologies that will be used for the development of the project. The technologies include Java as our programming language and Node-RED as our synchronization environment between the stages of the program.

3.2 Triangulation as a localization method

In trigonometry and geometry, triangulation is the process of determining the location of a point by forming triangles to it from known points [29]. Triangulation is of great importance, because it offers a way to determine one's location just by taking bearings to them.

A ***bearing*** (or azimuth) is the horizontal angle between the direction of an object and the North (or another object) [30]. The line that is marked based on the angle is called a ***bearing path***. In the current project we will be using ***absolute bearings,*** which refer to the clockwise angles between a point and the true North (true bearings). Bearings are usually given as a three-figure sequence. The sequence consists of two (2) characters and one (1) number: the first character is either N or S, referring to North or South respectively. Next part of the sequence is the angle value, usually in degrees (e.g., 45°). The last part of the sequence is either the characters E or W – referring to East or West respectively – and represents the direction of the angle away from the reference ray. The angle value will always be less than 90°. For instance, if point B is located exactly southeast of point A, the bearing from A to B is S45°E.

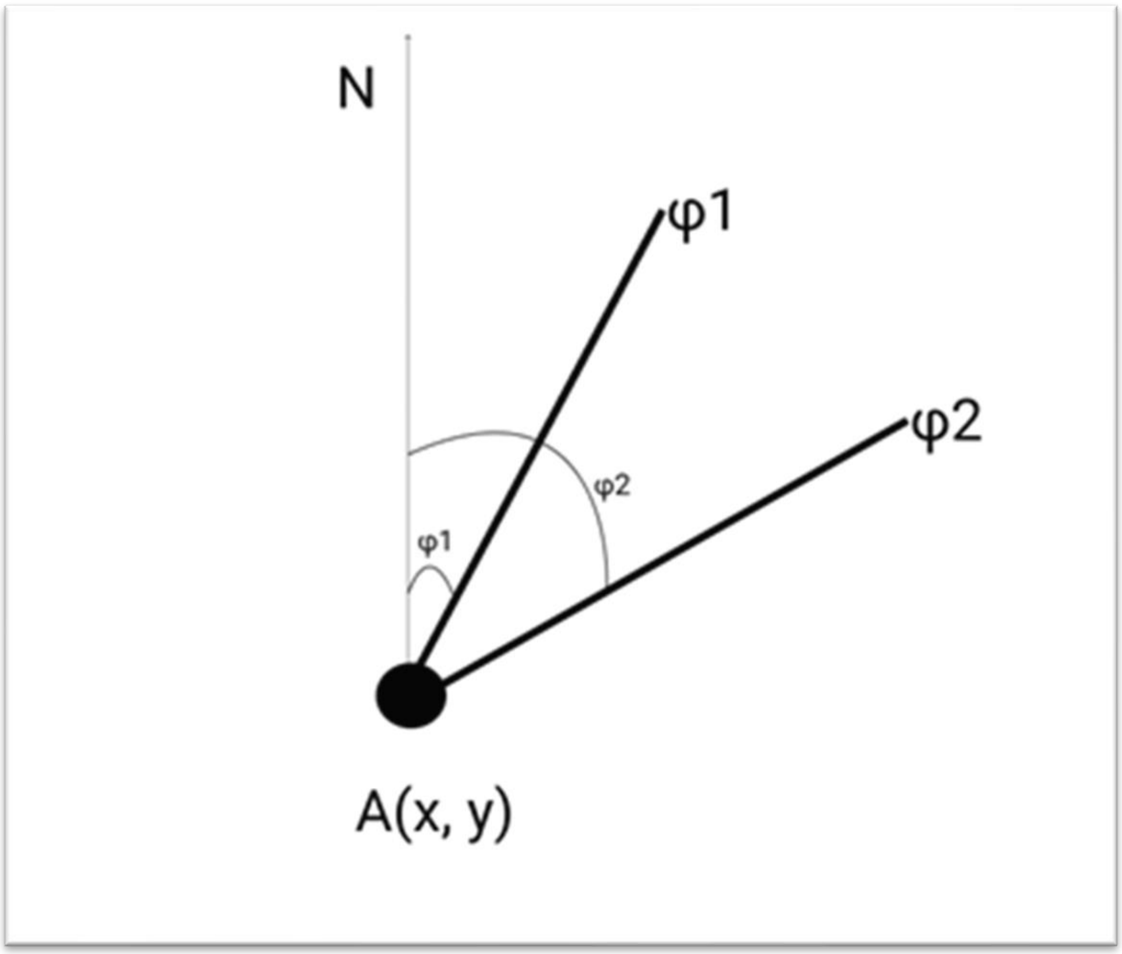


Figure 3.1. Sensor A with bearing angles φ1, φ2

Among its many uses, triangulation is widely used in many scientific fields, some of them including:

- Results confirmation/validation
- Bias minimization
- Data information validation

In the current chapter, however, we will look into triangulation as a method of localization, in order to determine a transmitter's position on the surface of the Earth. Moreover, in the next sections we will analyze the basic methodology that will be used, as well as general information about the concepts of triangulation.

3.3 Determination of all triangulation areas

Triangulation area is the result of the intersections of three (3) or more bearings by different sensors. It is defined as the common area of the intersections between the sensors' bearing paths. The triangulation area's size is analogous to the distance between the sensors and the intersection point; smaller triangulation areas are formed by sensors closer to the intersection point, in contrast to sensors far from the intersection point, where larger triangulation areas are formed (more on this matter at the end of section 3.6).

In order to determine all triangulation areas of more than one (1) transmitter, two sensors in the network are not enough. The main reason is that each sensor can only detect direction – according to its bearing angles – and not distance, preventing us from determining the exact geographical positions of the transmitters.

For instance, if a transmitter enters our area of interest (AOI), then two (2) sensors are enough to detect it, since both of them will bear to it and its geographical position will be the intersection point of their bearings. More analytically, the first sensor will bear to the transmitter but will not be able to detect its exact position, due to its inability to calculate the transmitter's distance from the sensor. For this reason, a second sensor needs to be placed in the network. The second sensor's bearing to the transmitter will intersect the first sensor's bearing at a specific geographical point. This point comprises the transmitter's exact location in our AOI.

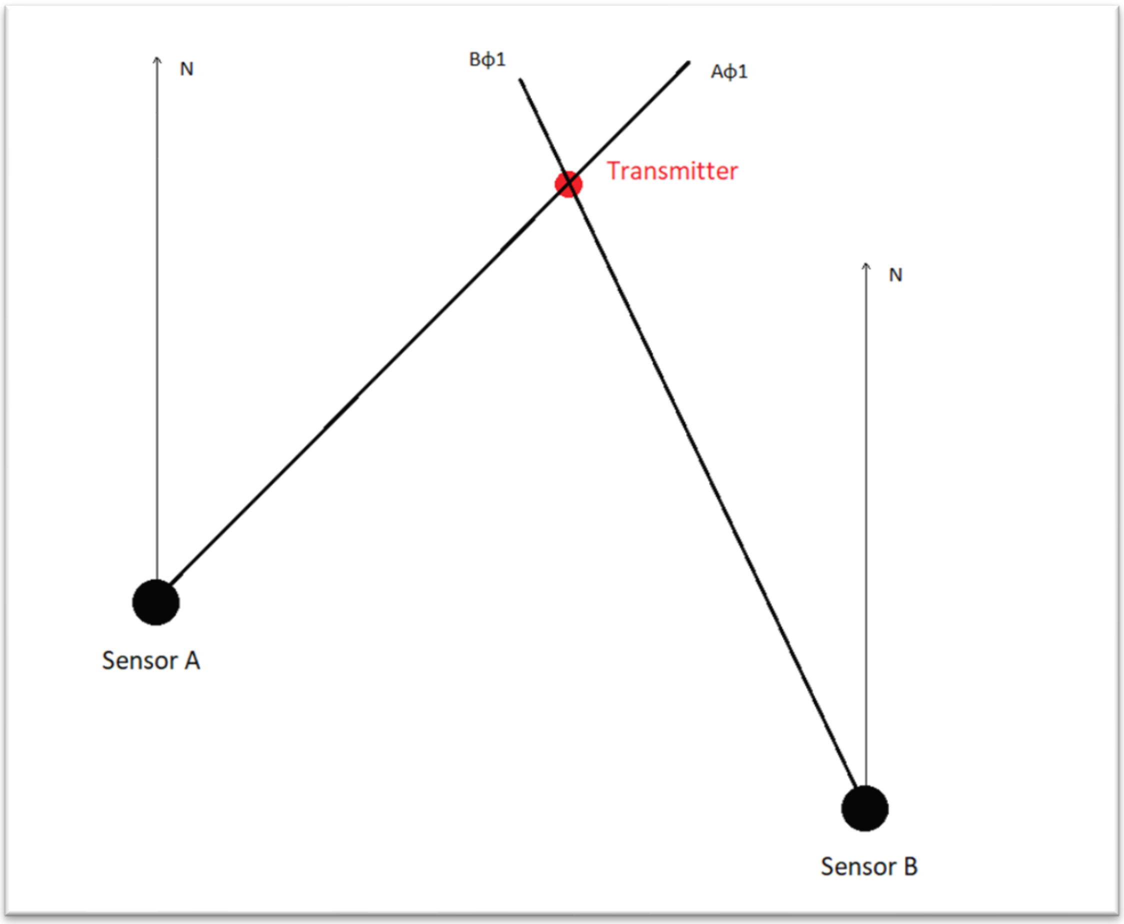


Figure 3.2. One transmitter is detected by two (2) sensors A and B

On the other hand, if two (2) or more transmitters enter our AOI, then two (2) sensors are not enough to detect the transmitters' positions. Following the above same procedure, the two sensors will bear to the transmitters' direction, each one acquiring two (2) bearing paths. As a result, we will now have 4 intersection points between the sensors' bearings (instead of two (2) as stated above), due to each sensor's two (2) different bearings to each transmitter. Since the intersection points are more than the transmitters (Figure 3.3 – marked with green dots), there is no way to know exactly which two (2) points correspond to the transmitters' positions.

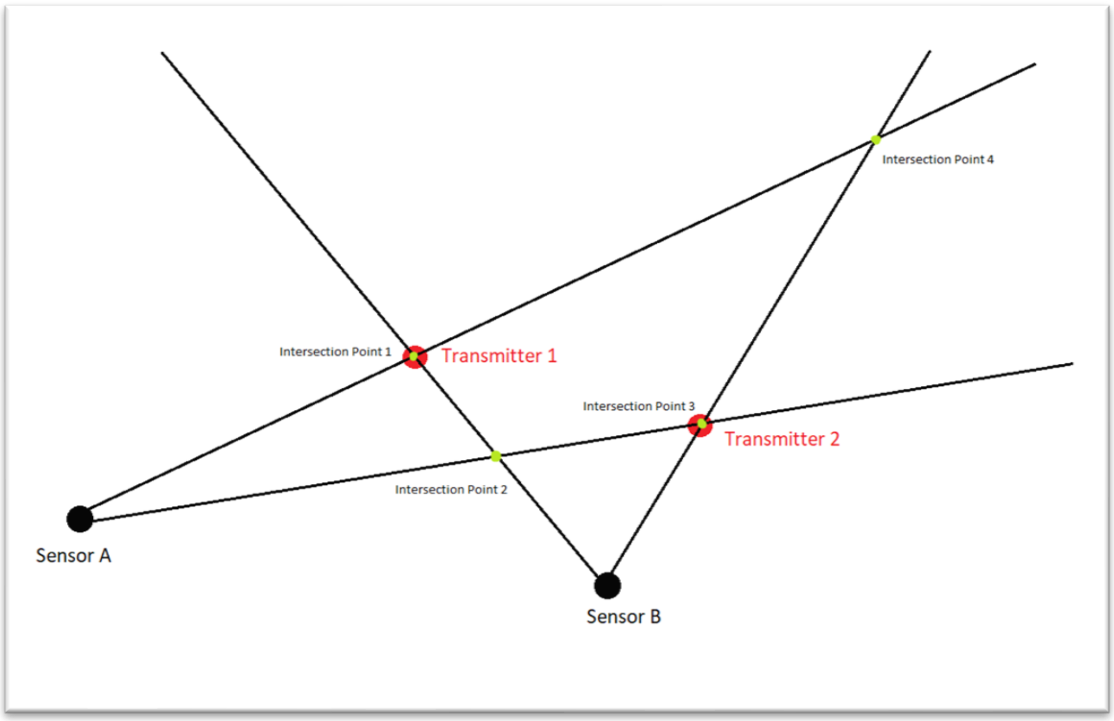


Figure 3.3. Two sensors are not able to detect two (2) or more transmitters

In these cases, one (1) or more sensors are placed in our network – depending on the area's requirements – estimating the exact location of each transmitter (Figure 3.4 – marked with blue dots).

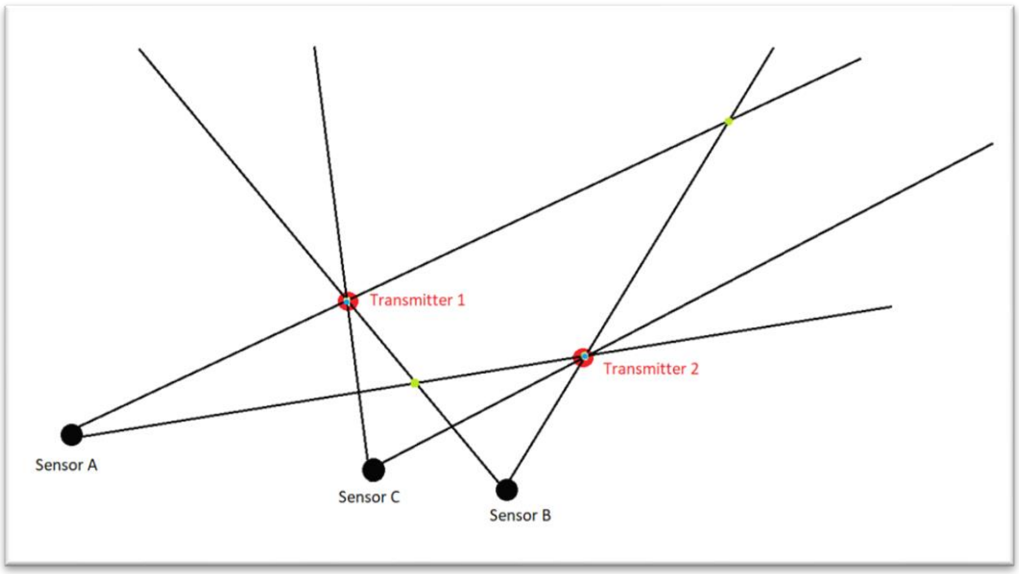


Figure 3.4. A third sensor is placed, in order to detect the two (2) transmitters

3.4 The importance of ±x degrees in bearing accuracy

When talking about triangulation and bearing, it is important to define each sensor’s bearing accuracy. Each sensor can detect a transmitter with a certain accuracy. Due to construction reasons, the detection accuracy may vary from 1° to 5°. This means that the bearing paths are not one (1) bearing path, but more like two (2), resembling the sides of a triangle, with the initial bearing being the triangle’s median side.

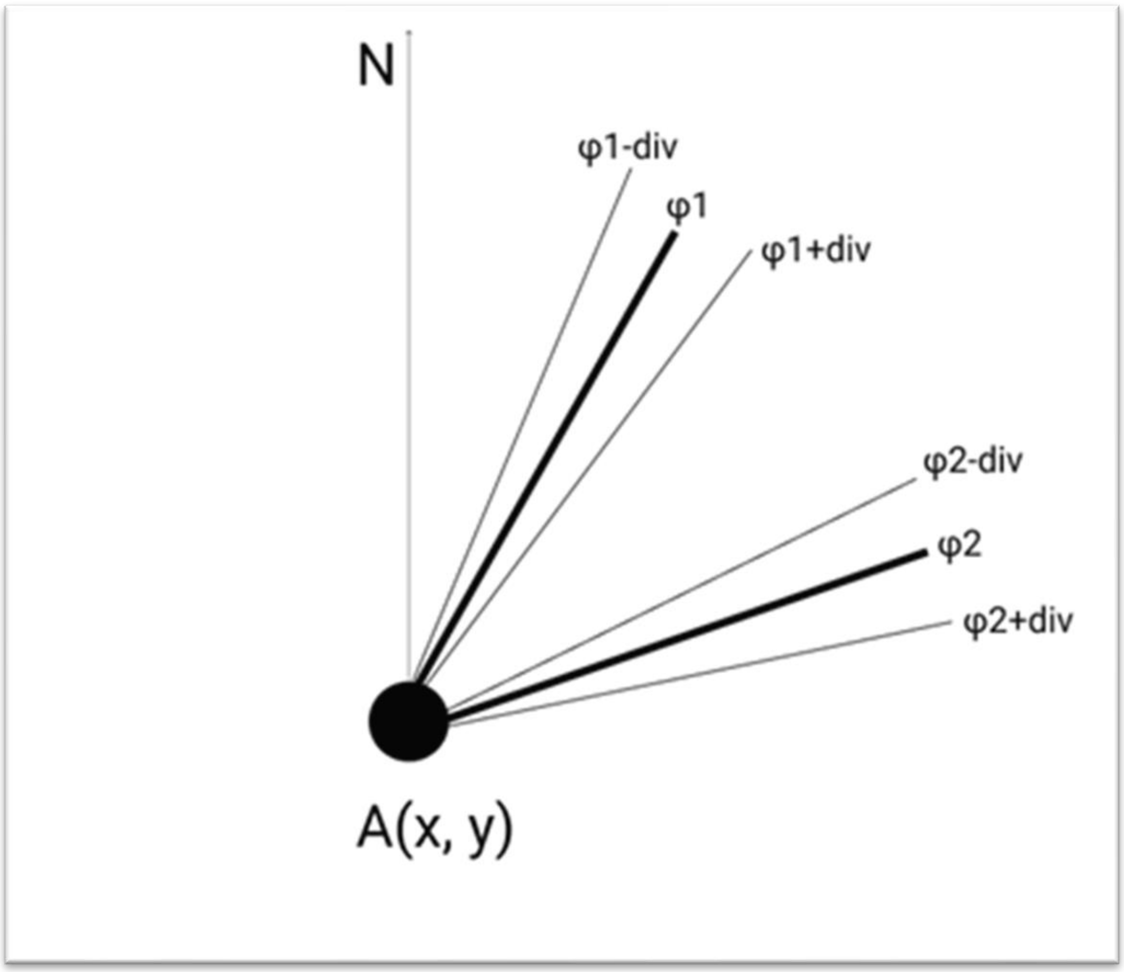


Figure 3.5. Sensor bearing angles with ±div

Therefore, a new variable is inserted into our problem, representing the divergence (error). The sensor's bearing accuracy is not defined by the initial bearing angle anymore, but is represented as a range, formed by the formula:

accuracy = [initial bearing angle – error, initial bearing angle + error] (I)

meaning that the sensor's bearing is not defined by a single angle, but by a range of angles from *(initial bearing-error)* to *(initial bearing+error)*. In this way, a wider range of angles, including a certain detection error, is covered. We will refer to the error as ***divergence (div)***. As stated above, the divergence may vary from 1° to 5°, according to parameters like environment, distance and other external factors. Using the above formula, we end up with two (2) bearing paths (initial-div, initial+div), instead of one (initial), that will be used in the triangulation method.

3.5 Bearing from multiple sensors for position estimation

Given two (2) sensors A (x1, y1) and B (x2, y2) in our network, along with their corresponding bearing angles Aφ1 and Bφ1, we use the accuracy formula **(I)** described above to provide each sensor with its two (2) bearing paths. After the application of the formula, sensor A's bearing accuracy ranges from (Aφ1-div) to (Aφ1+div), whereas sensor B's bearing accuracy ranges from (Bφ1-div) to (Bφ1+div). Combining the four (4) bearing paths creates the quadrilateral [Q1, Q2, Q3, Q4] that represents the common area between the two sensors' bearing paths.

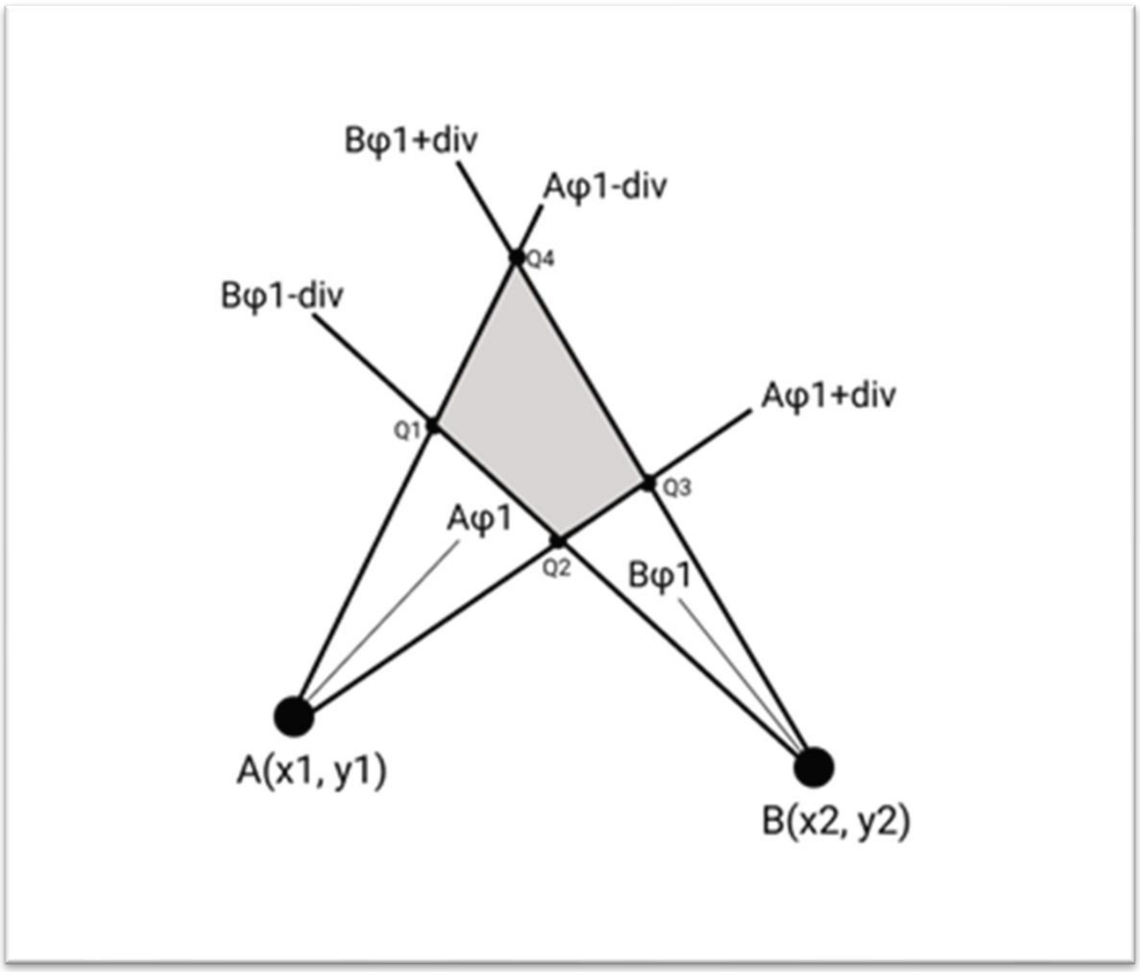


Figure 3.6. Quadrilateral representing the common area between bearings of sensors A, B

As mentioned earlier, in the case of two (2) or more transmitters, we need more than two (2) sensors in order to find all triangulation areas. Applying this method to three (3) random sensors A (x1, y1), B (x2, y2) and C (x3, y3) and their corresponding bearing angles Aφ1, Bφ1 and Cφ1, the image of our network is as follows:

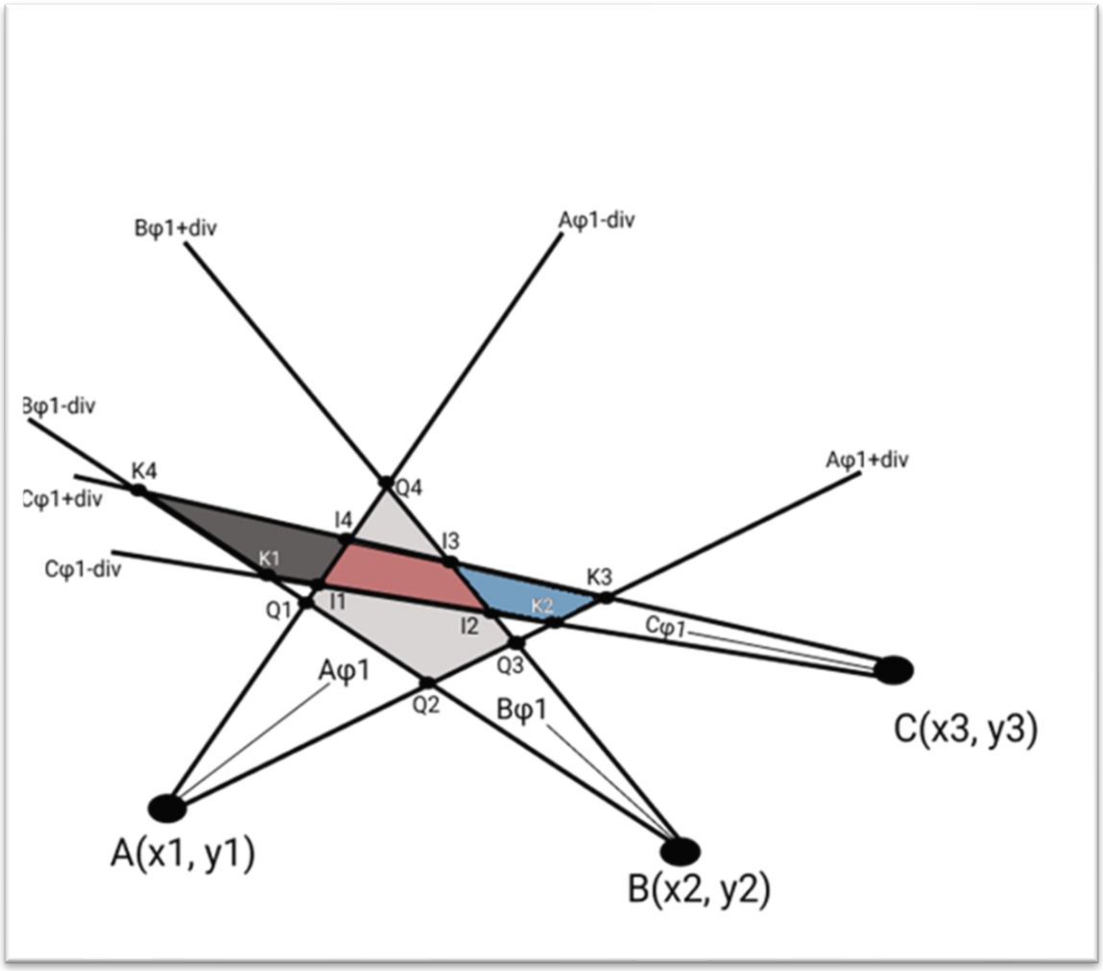


Figure 3.7. Triangulation area between the bearings of three sensors A, B and C[1]

As we can see, sensors A and B form the quadrilateral [Q1, Q2, Q3, Q4] (by the intersection of their bearing paths (Aφ1-div), (Aφ1+div), (Bφ1-div), (Bφ1+div)), whereas sensors B and C form the quadrilateral [K1, I2, I3, K4] (by the intersection of their bearing paths (Bφ1-div), (Bφ1+div), (Cφ1-div), (Cφ1+div)). On the other hand, sensors A and C form the quadrilateral [I1, K2, K3, I4] (by the intersection of their bearing paths (Aφ1-div), (Aφ1+div), (Cφ1-div) and (Cφ1+div)). By defining the three (3) quadrilaterals that are formed by the bearing paths of two (2) sensors each time, we can observe that they have common area with each other, defined by the newly

[1] All three (3) sensors use the same divergence value. The bearing paths for sensors A and B were drawn "wider", in order to achieve clarity and simplicity. In reality, they are as narrow as sensor C's bearing paths.

formed quadrilateral [I1, I2, I3, I4]. The new quadrilateral is the requested triangulation area, determined by the three (3) sensors A, B and C.

As stated in section 3.3, the triangulation area's size is analogous to the distance between the sensors and the intersection point; smaller triangulation areas are formed by sensors closer to the intersection point, in contrast to sensors far from the intersection point, where the formed triangulation areas are wider. By applying the accuracy formula **(I)**, we acquire two (2) bearing paths for each sensor and can take a notice at the formed triangulation areas below:

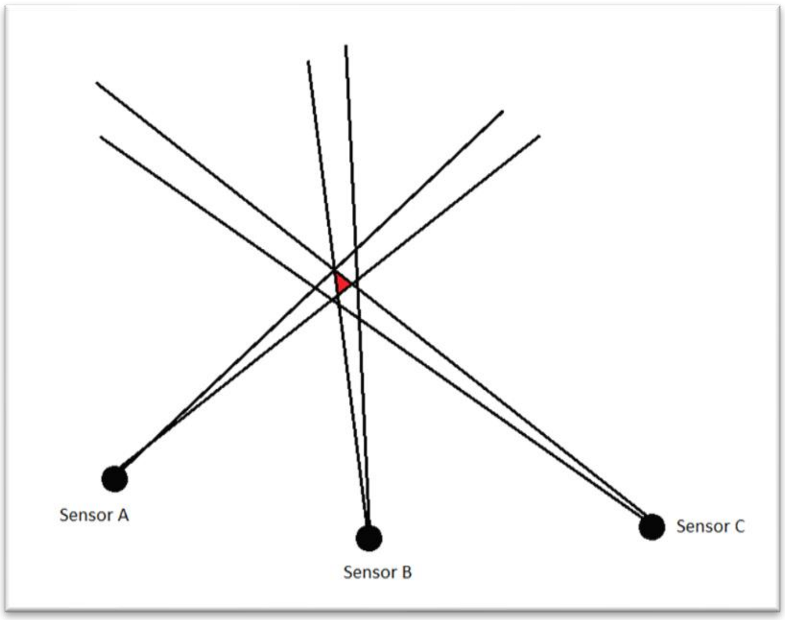


Figure 3.8. Small triangulation area by sensors closer to the intersection point

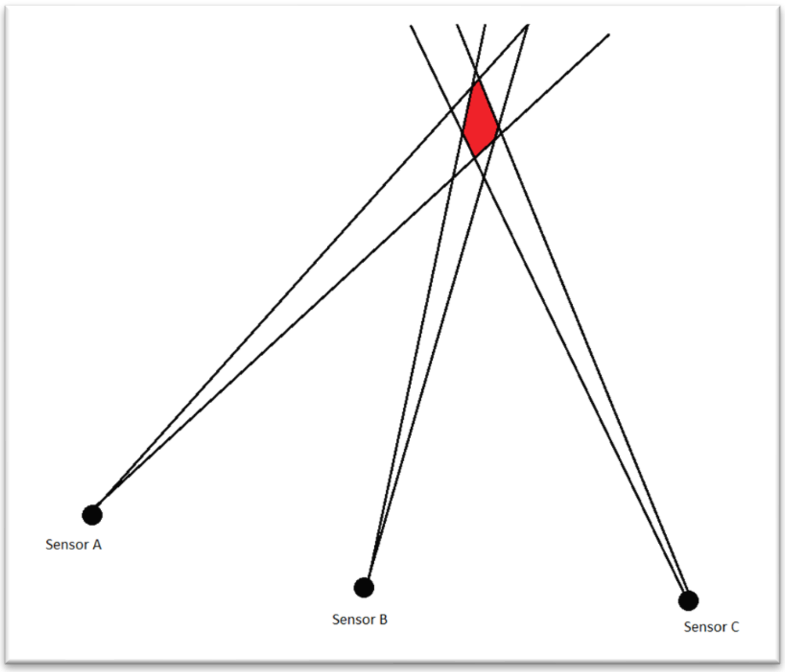


Figure 3.9. Wider triangulation area by sensors far from the intersection point

3.6 Sensor Description

The sensors that will be used in our network are a combination of an inertial system with the transmission of Very Low (VLF) and Ultra Low (ULF) electromagnetic frequencies [31]. The inertial system moves on a circular motion path and consists of four (4) units:

i. An electronic box containing:
   a. A signal generator system (producing electromagnetic signals)
   b. A force detection circuitry
ii. A telescopic antenna (from where the signals are emitted)
iii. A perpendicular axis (fixed underneath the generator box around which the circular motion will take place
iv. A base (through which the system will be able to rotate around its axis)

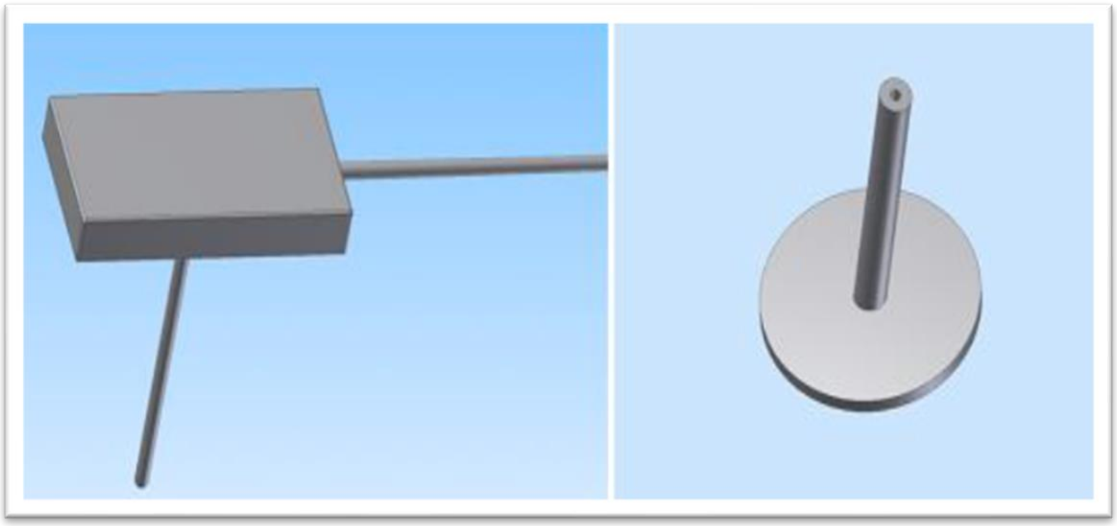


Figure 3.10. Inertial system with its base [31]

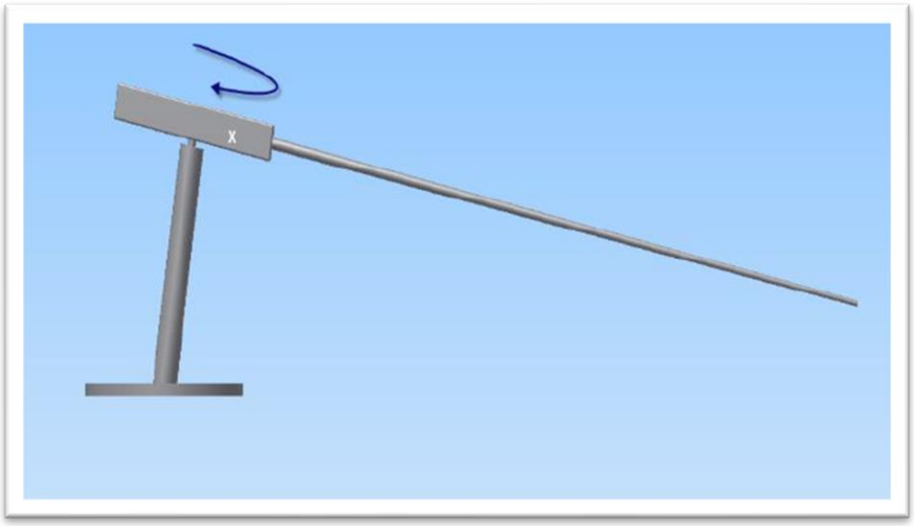


Figure 3.11. Inertial system with its four (4) units [32]

The system's motion is described by angular velocity ω(t). The weight of units **(i)** and **(ii)** causes a torque that makes the system rotate around its axis. The generator circuitry applies a signal on the antenna, which may differ (depending on the material that needs to be detected) and affect the movement. In the case of a detection – that is when a specific material is located towards the direction pointed by the antenna – a force is exerted on the sensor's antenna, altering its expected motion. The stronger the force, the sooner the system's movement will be affected. In other words, detection is translated as deceleration of the system.

The sensor's angle is calculated using Gray Code [33]–[37]. Gray Code is a binary enumerating system in which consecutive numbers differ by one (1) bit. For our system, a 9-bit Gray Code is used, in order to have $2^9 = 512$ angle combinations. A cycle is equal to 360°, so the accuracy of each step in the code is:

$$360° / 512 = 0.7031° \approx 0.7°$$

meaning that the minimum difference of the calculated angles is about 0.7°. The disk on which the Gray Code is printed is attached to an electronic reader communicating with a Raspberry Pi 4 board, which sends the angles' readings wirelessly to a computer. The readings are produced with a rate of approximate fifteen (15) angle readings per second, according to the sensor's manufacturing.

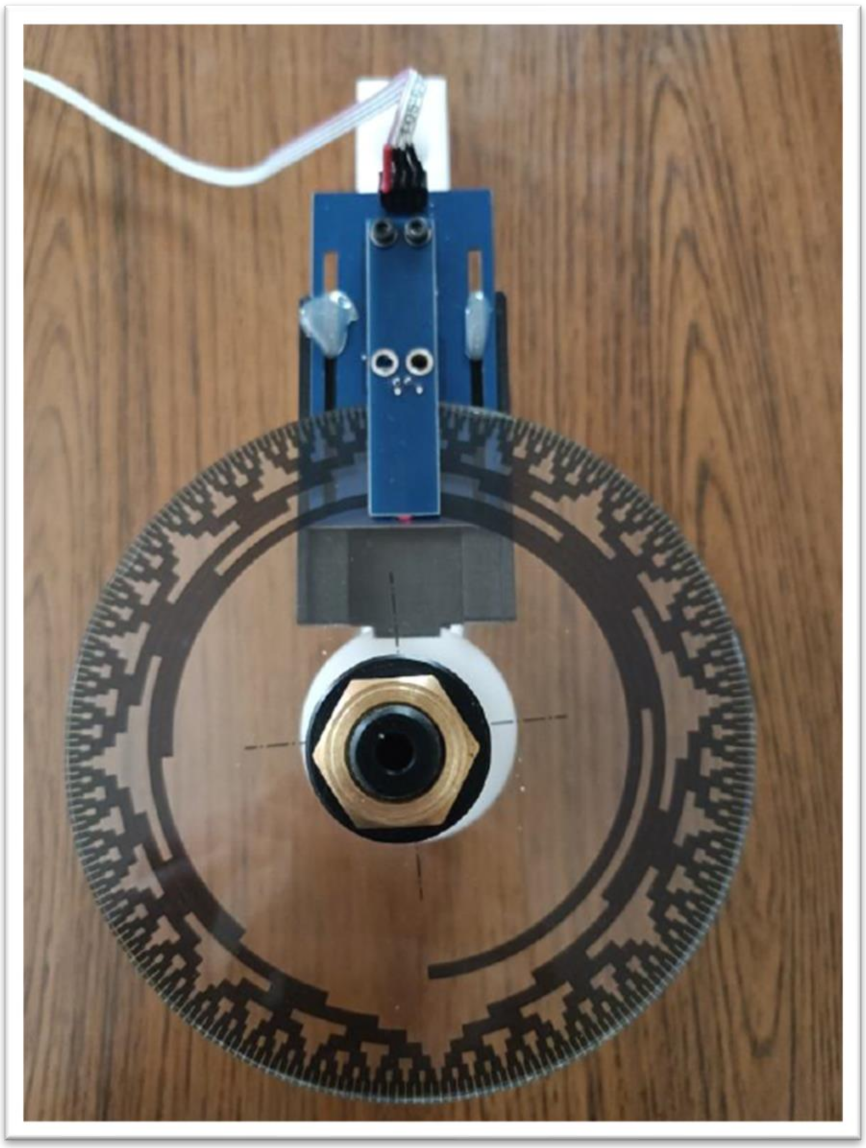


Figure 3.12. Vertical representation of the 9-bit Gray Code

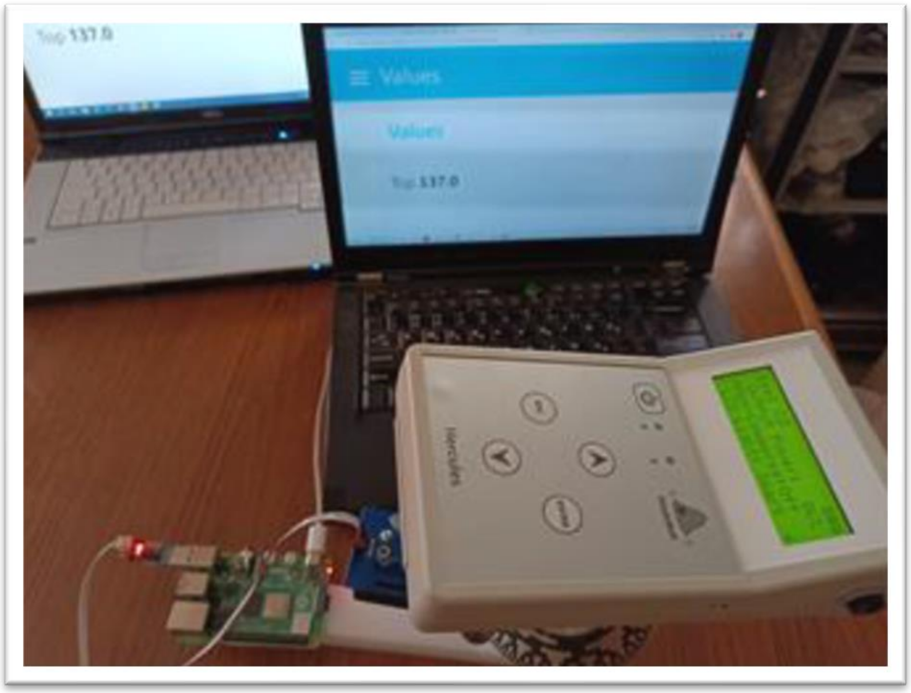


Figure 3.13. Complete inertial system

3.7 Technologies

3.7.1 Programming Language

The programming language that will be used for the development of the current project is Java [38]. The language decision was based on objective criteria – such as the numerous possibilities and libraries that Java offers (both in 2D and 3D) – as well as on the author's personal preferences. The JDK (Java Development Kit) version will be v19.0.2 and the IDE that will be used for the development will be IntelliJ v2021.2.3 Ultimate Edition by JetBrains [39].

3.7.2 Synchronization Environment – Node-RED

Node-RED is a flow-based development tool for visual programming, used for event-driven applications [40]. It provides a web browser-based flow editor and allows for code execution at the flow's nodes.

Our project consists of three (3) individual programs, which are serially executed and interdependent. This means that one program’s output is used as input for the next program. The three (3) programs are synchronized via Node-RED. The flow starts by executing the first program and acquiring its output. The output is then built into a new file that will be fed as input into the next program of the flow. The same procedure happens between the second and the third program as well. Synchronization, output acquisition and new input creation are executed in the intermediary nodes of the flow between the programs. More information about the programs and their inputs/outputs is provided in the next chapter.

Chapter 4 - Project Analysis

4.1 Chapter Overview – Problem Statement

The current chapter gives a detailed statement of the problem, the algorithms that were used for the purpose of the thesis and the experiment that was conducted, along with the acquired results.

There is a Wireless Sensor Network (WSN) organized over an area of several square kilometers, collecting data in real time. The data's nature is of field strength from various directions around each sensor. In order to efficiently organize and analyze the data and find high-intensity bearings (bearings at which a transmitter was detected), algorithms need to be developed. The existing sensors do not provide any information about the distance of a potential transmitter. It is important to be able to recognize new bearings at which a transmitter was detected and distinguish them from pre-existing ones, as well as how they change over time.

Furthermore, it is necessary to develop triangulation algorithms – in order to estimate the transmitter's position – and to project the transmitter's geographical position on a map, showing how it has changed over time.

4.2 Project Overview

The thesis consists of three (3) individual programs, all interdependent and executed in a serial way, in order to produce the final result. The flow of the programs is the following: each sensor rotates in a 360° radius collecting large amounts of data, which has to be filtered and organized in a way such, as to be used correctly and efficiently.

In order to be able to estimate a transmitter's position entering our area of interest (AOI), we have to be able to perform 3 actions:

i. Understand high-intensity bearings (meaning that "something" – a potential transmitter – might be in that area)
ii. Perform triangulation methods using these bearings, in order to find all triangulation areas
iii. Find the transmitter's position and project it on the map

Each of these actions are performed by the 3 individual programs referenced above and are named ***Bearing Definer***, ***Triangulation Detector*** and ***Centroid Detector*** respectively.

4.3 Algorithm Description

4.3.1 Bearing Definer

The first program of the flow is the ***Bearing Definer*** and acts as how its name implies: it is responsible for understanding/detecting all high-intensity bearings from the data collected and exporting them, in order to be used as input in the next step, the triangulation step.

As mentioned above, each sensor in our network rotates in a 360° radius collecting large amounts of data. The data consists of the current angle the sensor is bearing to and a timestamp for the corresponding entry. If a potential transmitter enters the AOI, the sensor's behavior changes as it displays a deceleration in its movement (Chapter 3 – section 3.6). This deceleration is not instantaneous, but it may last up to 2 seconds. The program's responsibility is to identify the bearings at which the deceleration took place and perform certain filtering and reducing algorithms, in order to efficiently make use of the information provided.

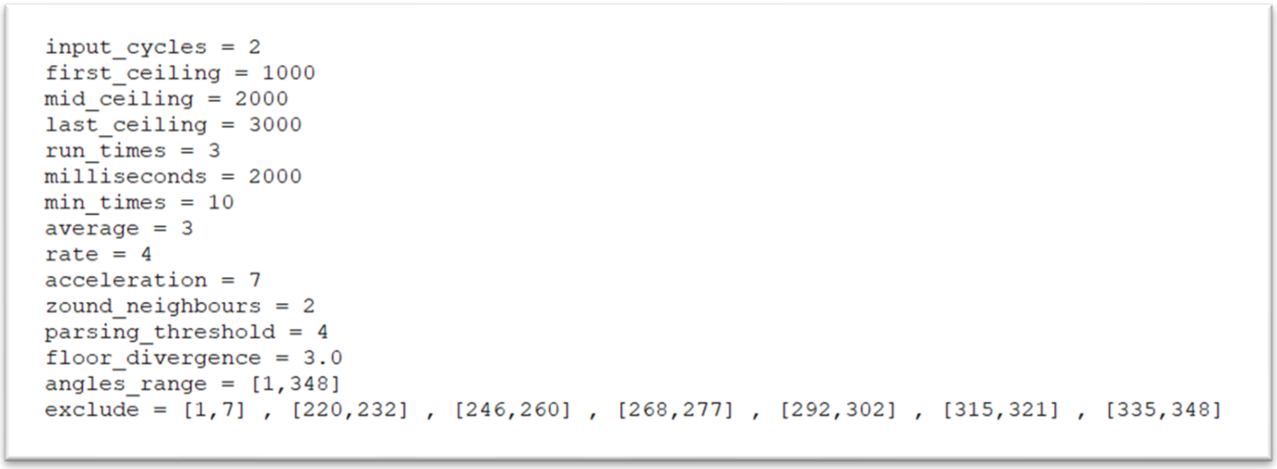
The program consists of six (6) stages:

i. Reading/Storing the data
ii. Grouping the data into averages/Computation of angular velocity-acceleration
iii. Production of "zound.txt"-"zound_with_two_neighbours.txt"/Application of "parsing_threshold"
iv. Sorting/Removing duplicate values
v. Exclusion of non-active ranges/Reducing
vi. Printing final output

The program begins by reading all necessary parameters from a file called *"parameters.txt"*. This file contains the following parameters:

- **input_cycles**: number of cycles to take into consideration when reading the data from the sensors
- **first_ceiling**: used as a starting point/base for the computation of a full cycle

- **mid_ceiling**: used as middle point for the computation of a full cycle
- **last_ceiling**: used as the last point for the computation of a full cycle
- **run_times**: number of times the program should run, before computing the final output
- **milliseconds**: amount of time in milliseconds – used to determine the duration for which the **“Main Thread”** should stay into ***“sleep”*** state (described in the next paragraph)
- **min_times**: minimum number of times of an appearance of an angle (used in the algorithm for input correction – described in the *General Notes* section – *(a) Correct Input Algorithm*, at the end of section 4.3.1)
- **average**: number of elements that will participate in the computation of the average (grouping of elements in the second stage)
- **rate**: units of time used, when computing the angular velocity/acceleration
- **acceleration**: the defining value of acceleration taken into consideration when producing the “zound.txt” file in the third stage
- **zound_neighbours**: number of adjacent angles taken into consideration when producing the “zound.txt” file
- **parsing_threshold**: used as floor in the third stage, in order to get the “winning” bearing angles
- **floor_divergence**: floor angle value used in the reducing algorithm in the fifth stage
- **angles_range**: range of angles that define our AOI
- **exclude**: range of angles excluded from our AOI



```
input_cycles = 2
first_ceiling = 1000
mid_ceiling = 2000
last_ceiling = 3000
run_times = 3
milliseconds = 2000
min_times = 10
average = 3
rate = 4
acceleration = 7
zound_neighbours = 2
parsing_threshold = 4
floor_divergence = 3.0
angles_range = [1,348]
exclude = [1,7] , [220,232] , [246,260] , [268,277] , [292,302] , [315,321] , [335,348]
```

Figure 4.1. “parameters.txt” file

The algorithm consists of a thread that is initialized at the beginning of the program and runs infinitely. The thread – called “**Main Thread**” – is responsible for all the work executed. It can go into two (2) states: *“**sleep**”* and *“**awake**”*. The thread stays in *“sleep”* state for n milliseconds, as defined by the corresponding parameter “milliseconds” in the “parameters.txt” file. After n milliseconds, it goes into *“awake”* state, in which it wakes up and searches to read an input file called “input_for_bearing.dat” in the existing directory. If no input file is found, the thread goes into *“sleep”* state. If an input file is found, the thread proceeds to start with the execution of the main body of the algorithm.

4.3.1.1 Stage 1: Reading/Storing the data

The algorithm begins by reading data from an input file called “input_for_bearing.dat”. Each line in the input file consists of information about the bearing angle and a timestamp at which the specific data was collected. The input file is basically the output data of the sensor.

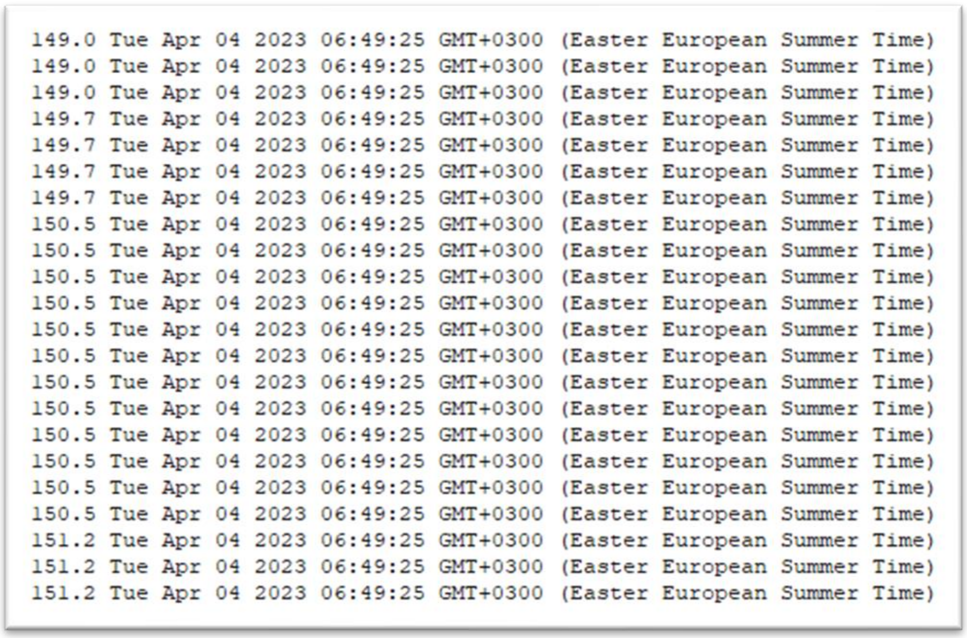

```
149.0 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
149.0 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
149.0 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
149.7 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
149.7 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
149.7 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
149.7 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
150.5 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
150.5 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
150.5 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
150.5 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
150.5 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
150.5 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
150.5 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
150.5 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
150.5 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
150.5 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
150.5 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
150.5 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
151.2 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
151.2 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
151.2 Tue Apr 04 2023 06:49:25 GMT+0300 (Easter European Summer Time)
```

Figure 4.2. Input file "input_for_bearing.dat"

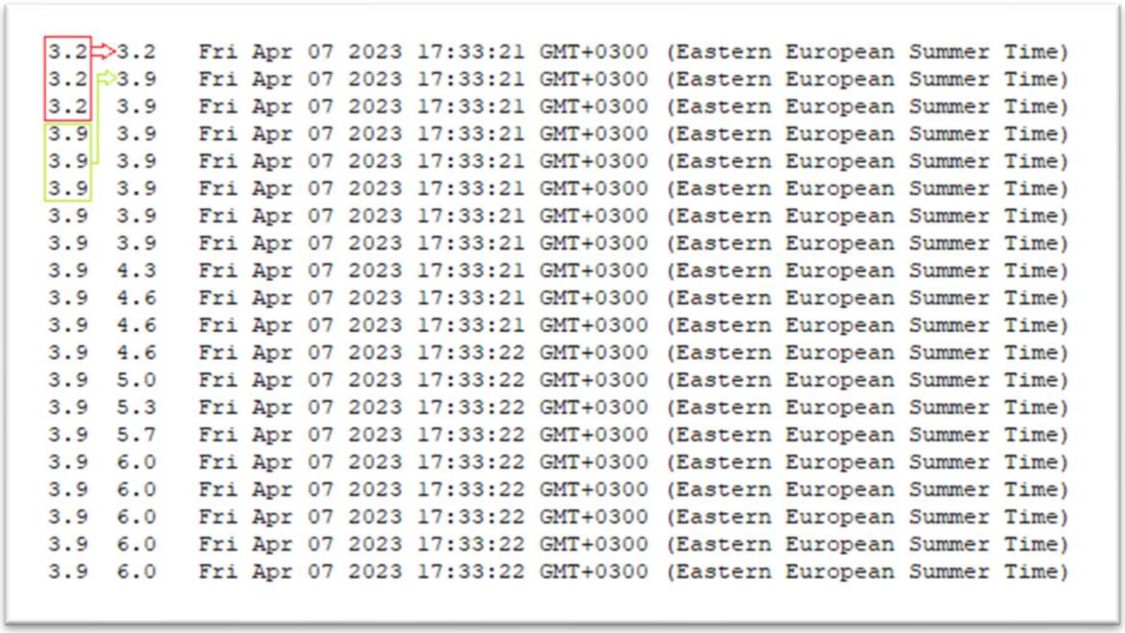
Each line is stored in a list for future computational purposes. The number of lines read (thus the amount of data) depends on the “input_cycles” parameter, as specified in the “parameters.txt” file. For instance, for *input_cycles*=2, the algorithm stops reading data when completing two (2) full cycles.

Each cycle is computed based on the values of the bearing angles and the “first_ceiling”, “second_ceiling” and “third_ceiling” parameters, as specified in the “parameters.txt” file. The computation of a full cycle uses the following procedure: three (3) Boolean variables are used, all initialized with *false*. Each variable is set to *true* whenever the bearing angle falls between the specified ceiling parameter and the sum of the ceiling parameter plus a fixed divergence number (for simplicity reasons, in the current project the fixed value is equal to 10). For instance, let pi, pm and pl be the Boolean variables initialized with *false* and “first_ceiling”=100, “second_ceiling”=200 and “third_ceiling”=300. When traversing the input data, if the bearing angle falls between 100 and 110, pi is set to *true*. The same method is applied for the other two variables. When all three variables are *true* (hence we started from angle≈100 and reached angle≈300), a full cycle is completed. The reading/storing of the data stops when the number of cycles reaches the number of “input_cycles” specified in the “parameters.txt” file.

4.3.1.2 Stage 2: Grouping the data into averages – Computation of angular velocity/acceleration

4.3.1.2.(a) Grouping the data into averages

Due to its high sampling rate, the sensor produces a large amount of data. Each angle appears numerous continuous times in the data, as the sensor produces this data at one unit of time. This happens due to the sensor's nature/manufacturing (as stated in Chapter 3 – section 3.6, the sensor produces approximately about 15 angle readings per second). In order to be able to have a more accurate insight of the data and be able to use it more effectively, we need to find a way to group it. For this purpose, we will group the data into groups of averages. Each group of averages will take into consideration "average" number of elements – as described in "parameters.txt" – and will be reduced into one final value, the average value of the elements participating in the group. For instance, for "average"=3, 3 lines from the input file will be used into the grouping method. The bearing angles of each line from the input file will be accumulated and then divided by 3 (according to the "average" parameter), in order for the 3 bearing angles to be reduced into one final angle. We repeat the process for all angles in the "input_cycles" range and a new file is produced. The new file is called "average.txt" and contains all reduced average bearing angles, along with their timestamps. The "average.txt" will be used for the computations below.



```
3.2  3.2   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.2  3.9   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.2  3.9   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  3.9   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  3.9   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  3.9   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  3.9   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  3.9   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  4.3   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  4.6   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  4.6   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  4.6   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  5.0   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  5.3   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  5.7   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  6.0   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  6.0   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  6.0   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  6.0   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  6.0   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
```

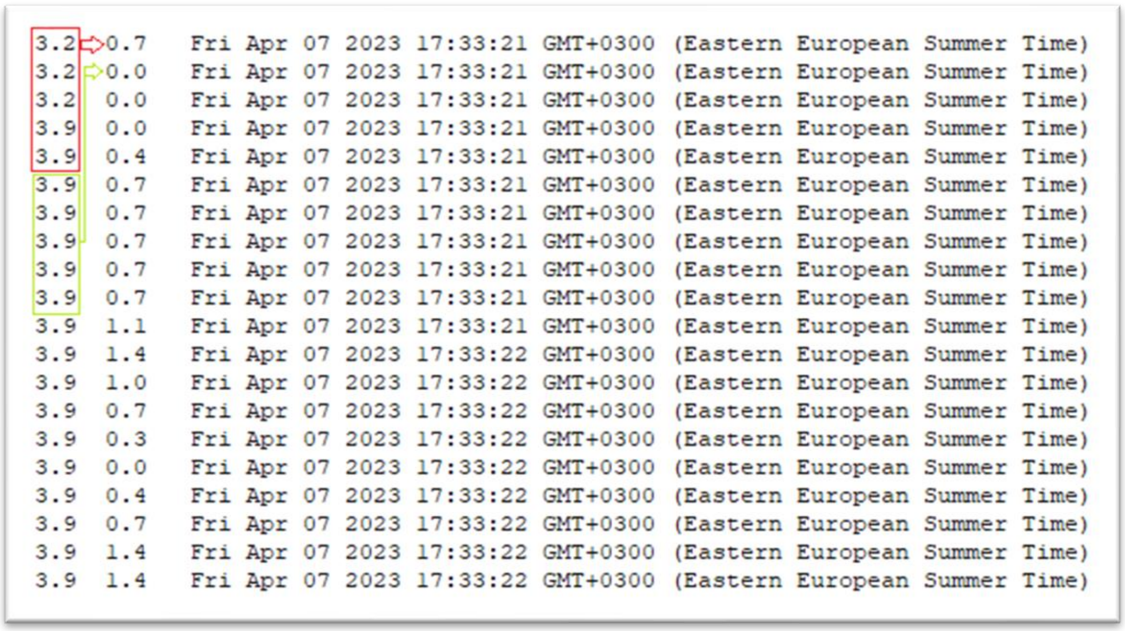
Figure 4.3. Averages file "average.txt" with "avg"=3

4.3.1.2.(b) Computation of angular velocity/acceleration

As stated earlier, each sensor in our network rotates in a 360° radius collecting large amounts of data. Therefore, it develops angular velocity and thus, angular acceleration. Each time the sensor detects "something", its antenna faces a slight deceleration. We are interested in the values of this deceleration, as the deceleration is the result of interaction between the sensor and the potential transmitter. In other words, deceleration indicates detection (Chapter 3 – section 3.6).

In order to find the values of this deceleration, we first need to compute the sensor's angular velocity. We will compute both angular velocity and angular acceleration by finding the rate of change of the respective values per unit of time. Using the "rate" parameter, we group the data produced in the "average.txt" file per ("rate"+1) number of elements (e.g. for "rate"=4, each group of data includes the starting angle plus 4 more angle readings – in total five (5) angle readings) and compute the difference between the last and the first value (difference of bearing angles for velocity, difference of velocity values for acceleration).

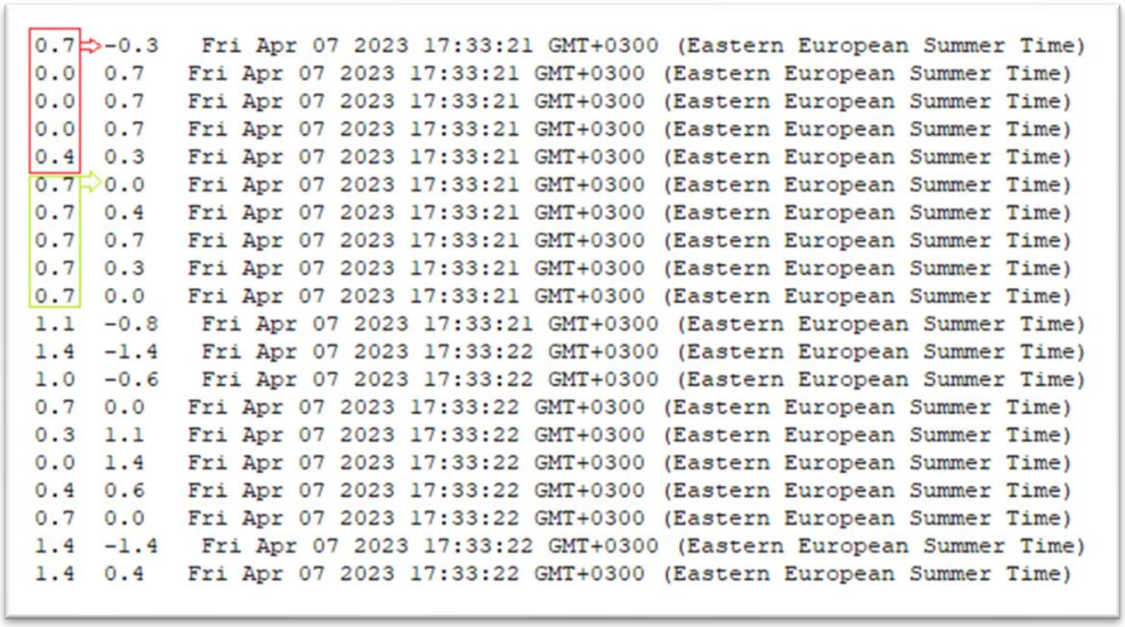
Starting with the velocity, we compute the difference of the bearing angles column (per 2) in the "average.txt" file. We start by traversing the angles column – line per line – in the "average.txt" file (per 2) and subtract the last value in the group from the first one. This produces a new value for the current group of elements, which corresponds to the sensor's angular velocity at the specific timestamp.



```
3.2  0.7   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.2  0.0   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.2  0.0   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  0.0   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  0.4   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  0.7   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  0.7   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  0.7   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  0.7   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  0.7   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  1.1   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
3.9  1.4   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  1.0   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  0.7   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  0.3   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  0.0   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  0.4   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  0.7   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  1.4   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
3.9  1.4   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
```

Figure 4.4. Angular velocities with "rate"=4

We continue the process until the end of the data and we are left with a file so long as the "average.txt" file, which contains the angular velocities of the sensor per unit of time. By repeating the same process, this time though on the new file's column of velocities (instead of the angle bearings), we will find the rate of change of the velocity per unit of time, which corresponds to the sensor's angular acceleration at the specific timestamp.



```
0.7 -0.3   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
0.0  0.7   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
0.0  0.7   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
0.0  0.7   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
0.4  0.3   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
0.7  0.0   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
0.7  0.4   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
0.7  0.7   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
0.7  0.3   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
0.7  0.0   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
1.1 -0.8   Fri Apr 07 2023 17:33:21 GMT+0300 (Eastern European Summer Time)
1.4 -1.4   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
1.0 -0.6   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
0.7  0.0   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
0.3  1.1   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
0.0  1.4   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
0.4  0.6   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
0.7  0.0   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
1.4 -1.4   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
1.4  0.4   Fri Apr 07 2023 17:33:22 GMT+0300 (Eastern European Summer Time)
```

Figure 4.5. Angular accelerations with "rate"=4

By completing the process, we have a new file consisting of four (4) columns: the bearing angles, the angular velocity, the angular acceleration and the corresponding timestamp. In order to keep the program "clear" and not produce a huge amount of useless files/data, we overwrite the new file on the "average.txt" and will use in the next step to produce the "zound.txt" file.

4.3.1.3 Stage 3: Production of "zound.txt" / "zound_with_two_neighbours.txt" / Application of "parsing_threshold"

4.3.1.3.(a) Production of "zound.txt"

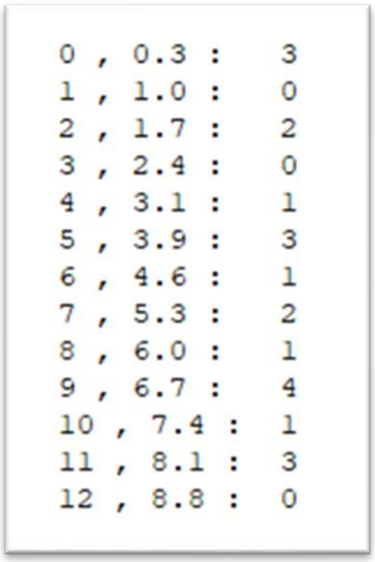
The "zound.txt" file is of prime importance to the program, as it contains the number of times an acceleration value appeared for a specific angle, along with the corresponding angle. The acceleration value depends on the "acceleration" parameter. After the computation of the angular velocities and accelerations, the next step is to apply a filtering technique on the data. As we mentioned earlier, we are interested in the sensor's deceleration when detecting "something". Deceleration is translated as a negative acceleration value. Since we have all acceleration values of the sensor (produced in the previous step), we can now filter only the values we are interested in. For this purpose, we use the “acceleration” parameter in the “parameters.txt” file. The “acceleration” parameter acts as a ceiling threshold, allowing us to obtain only acceleration values less-than-or-equal to minus ”acceleration” (since we are interested in negative acceleration values, thus deceleration values).

We begin by traversing the “average.txt” file, taking into consideration only elements belonging to the accelerations’ column. We accept acceleration values less-than-or-equal to minus “acceleration” and greater-than-or-equal to a constant floor value (for simplicity, our floor value is -10). The formula for an acceptable acceleration value is:

-10 <= acceleration value <= -“acceleration” parameter (II)

For instance, if “acceleration”=0.7, we want to filter all acceleration values that fall in the range [-10, -0.7]. Any acceleration value falling in this range indicates that the deceleration is “strong” enough to be accepted and that the sensor is in fact detecting “something”.

Each acceptable acceleration value is stored in the “zound.txt” file, along with its corresponding bearing angle. An example of the “zound.txt” file is presented below:



```
0 , 0.3 :   3
1 , 1.0 :   0
2 , 1.7 :   2
3 , 2.4 :   0
4 , 3.1 :   1
5 , 3.9 :   3
6 , 4.6 :   1
7 , 5.3 :   2
8 , 6.0 :   1
9 , 6.7 :   4
10 , 7.4 :  1
11 , 8.1 :  3
12 , 8.8 :  0
```

Figure 4.6. File "zound.txt"

According to the "zound.txt" provided, at bearing angle 7.4°, the number of acceleration values acceptable according to the formula above is one (1). This means that when bearing at 7.1°, the sensor spotted an acceptable deceleration value one (1) times.

Due to construction reasons, at each detection the sensor has an error tolerance of approximately two (2) steps. This means that in the first cycle, a detection may occur at a certain angle (e.g., 20°), whereas in the next cycle the same detection may occur at an earlier angle (e.g., 18°) or a subsequent angle (e.g., 22°). In order to be accurate, we need to take into consideration all neighboring acceleration values when forming the final result. The error tolerance is referenced as *divergence* and is described in Chapter 3 – section 3.4.

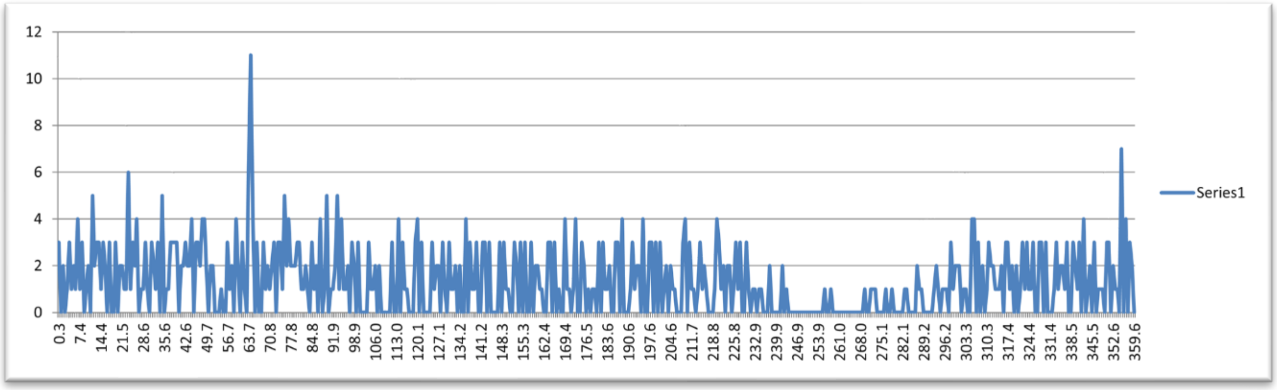


Figure 4.7. "zound.txt" chart

In the chart above, we can observe that a detection was performed at circa 62°. The event did not happen instantaneously, but lasted for some time, affecting neighboring angle values as well. For this reason, we need to take neighboring acceleration values into account, too, in order to consider the error tolerance when forming the final result.

4.3.1.3.(b) Production of "zound_with_two_neighbours.txt"

A new file is produced from "zound.txt" and is called "zound_with_two_neighbours.txt".

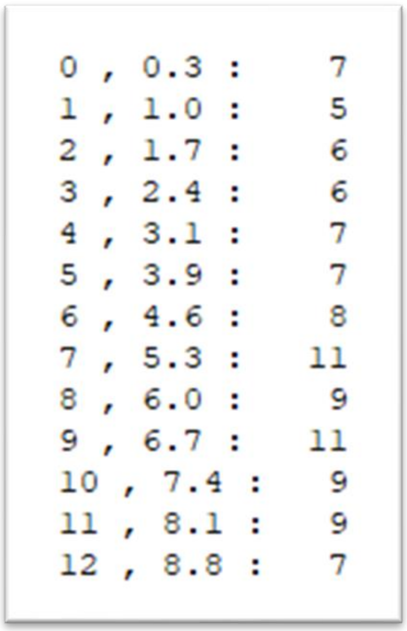

```
0 , 0.3 :    7
1 , 1.0 :    5
2 , 1.7 :    6
3 , 2.4 :    6
4 , 3.1 :    7
5 , 3.9 :    7
6 , 4.6 :    8
7 , 5.3 :   11
8 , 6.0 :    9
9 , 6.7 :   11
10 , 7.4 :   9
11 , 8.1 :   9
12 , 8.8 :   7
```

Figure 4.8. File "zound_with_two_neighbours.txt"

In order to produce the “zound_with_two_neighbours.txt” file, we traverse the “zound.txt” file and for each acceleration value, we add the two top and two bottom neighboring acceleration values. For the above “zound.txt” file”:

6 , 4.6 :	1
7 , 5.3 :	2
8 , 6.0 :	**1**
9 , 6.7 :	4
10 , 7.4 :	1
11 , 8.1 :	3
12 , 8.8 :	0

the “zound_with_two_neighbours.txt” file for the acceleration value at angle 6.0° (index 8) will consist of the accelerations’ sum at angles 4.6°, 5.3°, 6.0°, 6.7° and 7.4° (indices 6, 7, 8, 9 and 10). More specifically, the new acceleration value will be the acceleration at angle 6.0° plus its two (2) neighbors up (4.6°, 5.3°) and down (6.7°, 7.4°):

6 , 4.6 :	8
7 , 5.3 :	11
8 , 6.0 :	***9***
9 , 6.7 :	11
10 , 7.4 :	9
11 , 8.1 :	9
12 , 8.8 :	7

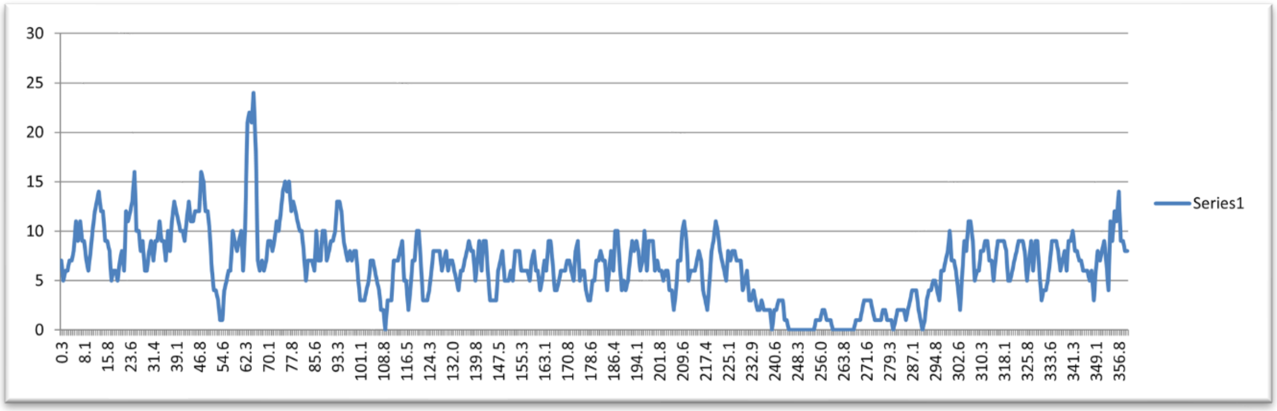


Figure 4.9. “zound_with_two_neighbours.txt” chart

By observing the chart in Figure 4.9, we can see that the deceleration event was captured more smoothly after computing the “zound_with_two_neighbours.txt”. The “winning” angles are more distinct and offer a more accurate perception of the event. The newly created file is like passing a filter to the already “harsh” file “zound.txt”, which makes the study of the results more efficient.

4.3.1.3.(c) Application of "parsing_threshold"

After producing the new file "zound_with_two_neighbours.txt", it is time to use the "parsing_threshold" parameter, in order to only choose certain values of acceleration. We traverse the "zound_with_two_neighbours.txt" file and store only acceleration values that are greater-than-or-equal to the "parsing_threshold". In this way, we filter all winning angles we deem worthy of keeping.

4.3.1.4 Stage 4: Sort/Remove duplicates

Upon completion of stage 3, we have acquired all winning angles, according to the "parsing_threshold" parameter. Since many "zound_with_two_neighbours.txt" are produced, due to the number of times the program is executed (based on the "run_times" parameter), we end up with a large amount of duplicate angle values. Naturally, the next step will be to remove all duplicate values.

We begin by sorting the list with all winning angles in ascending order, taking advantage of the Collections.sort() function provided by Java. After sorting the angles list, we remove all duplicate values by passing the angles in a HashSet. The main characteristic of a HashSet is that each item is unique, so at the insertion of each angle into the HashSet the check to see if the angle already exists in the current HashSet is performed. All checks are performed by Java, so no extra code is required. Finally, we copy all sorted unique angle values back to our main list and end up with a list of winning angles sorted in ascending order and unique, ready to be fed to the next stage.

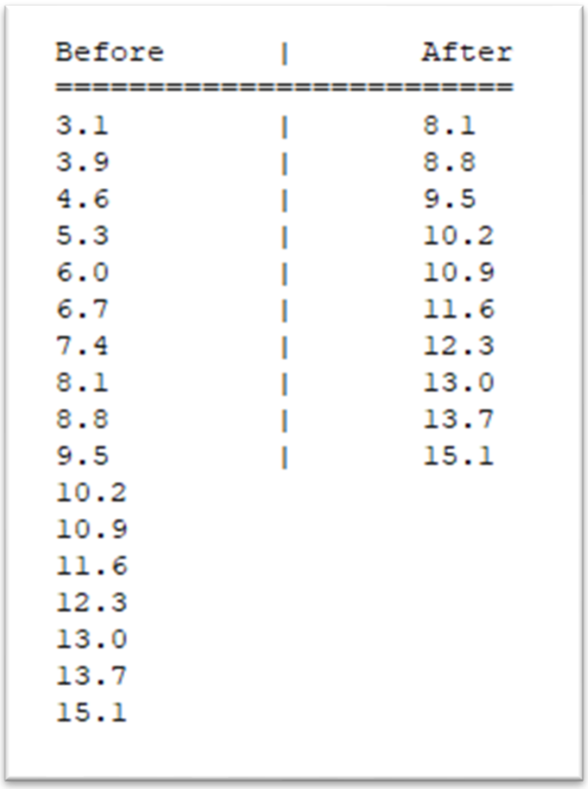
4.3.1.5 Stage 5: Exclusion of non-active ranges/Reduction

In the current stage, all winning angles from the previous stage must be checked and reduced even further, in order to end up with a much clearer output.

4.3.1.5.(a) Exclusion of non-active ranges

The list with the winning angles contains all angles in the range [0, 360]. Not all angles are important for the final output, though. In the “parameters.txt” file, we have described two (2) parameters, the “angles_range” and the "exclude”. The “angles_range” corresponds to the range at which our AOI exists (depends solely on each sensor’s bearing range), whereas the “exclude” describes the angle intervals that we want to exclude. For instance, if “angles_range”=[0, 360] and “excludes”=[50, 60] , [85, 90], then our active range of interest is split into three (3) ranges: [0, 49], [61, 84] and [91, 360].

For this purpose, we need to keep only winning angle values falling into either of these three (3) active ranges and exclude all angle values that fall outside of our active range of interest. At the screenshot below, with “angles_range”=[1, 348] and “exclude”=[1, 7], the final angles both before and after the exclusion of non-active ranges are depicted:



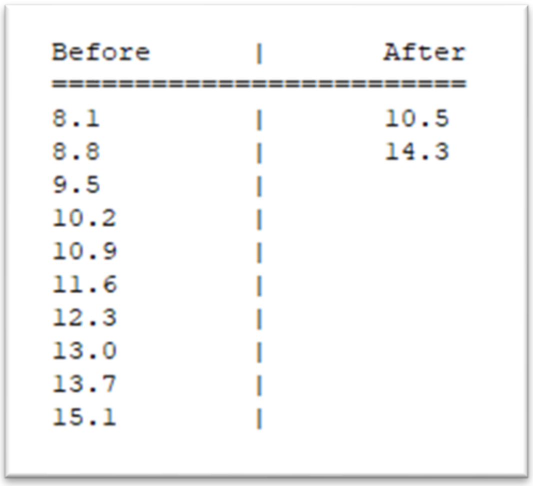
```
Before        |        After
==========================
3.1           |        8.1
3.9           |        8.8
4.6           |        9.5
5.3           |        10.2
6.0           |        10.9
6.7           |        11.6
7.4           |        12.3
8.1           |        13.0
8.8           |        13.7
9.5           |        15.1
10.2
10.9
11.6
12.3
13.0
13.7
15.1
```

Figure 4.10. Before/After the exclusion of non-active ranges

4.3.1.5.(b) Reduction

After the exclusion of non-active ranges, the final output of the first program is almost ready. By studying the remaining angles, we observe that some angle values are very close to each other, differing even less than 1°. For this reason, we perform a specific reduction algorithm, in order to group all these neighboring angles into one (1).

We begin by defining a “floor” variable that points to the first angle of the list. We then traverse the list of angles by comparing the “floor” to each angle of the list. Our aim is to group all angles that differ at most floor + “floor_divergence” (as defined in “parameters.txt”) and find their mean average. If an angle falls in the range of floor + “floor_divergence”, we add it to the group of angles for reduction. If an angle does not fall in the above range, we compute the mean average of the group of angles until that angle, reduce them into one (1) and repeat the procedure for the current angle and its successors. Following this method, the resulting angles list is reduced by at least 3/4 of the initial non-reduced list. The resulting angles are the official output of the first part of the program. At the screenshot below, the angles both before and after the reduction algorithm are depicted:



```
Before      |       After
=========================
8.1         |       10.5
8.8         |       14.3
9.5         |
10.2        |
10.9        |
11.6        |
12.3        |
13.0        |
13.7        |
15.1        |
```

Figure 4.11. Before/after the reduction algorithm

4.3.1.6 Stage 6: Print final results

In the sixth and final stage, the remaining reduced angles are printed. These angles are the final output of the ***Bearing Definer*** program and will be used as input to the next program, the ***Triangulation Detector***. They comprise the bearing angles at which a detection was performed and will be used to perform triangulation.

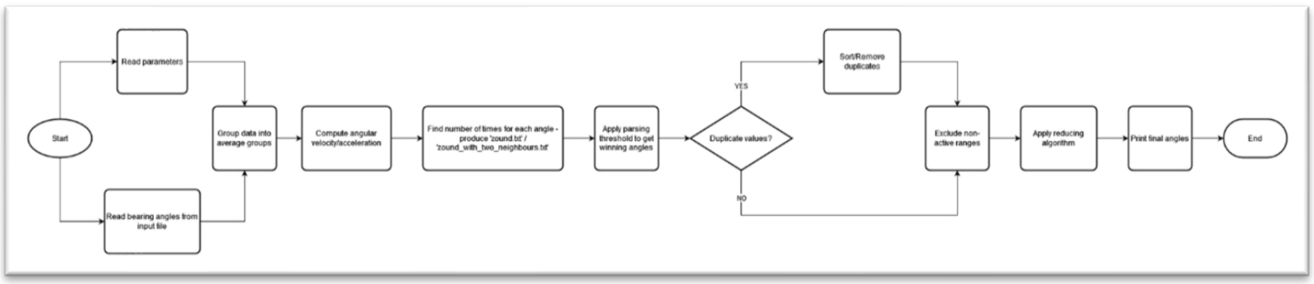

Start
Read parameters
Read bearing angles from input file
Group data into average groups
Compute angular velocity/acceleration
Find number of times for each angle - produce 'zound.txt' / 'zound_with_two_neighbours.txt'
Apply parsing threshold to get winning angles
Duplicate values?
YES
NO
Sort/Remove duplicates
Exclude non-active ranges
Apply reducing algorithm
Print final angles
End

Figure 4.12. *Bearing Definer* flowchart

General Notes

During the development of the *Bearing Definer,* two (2) main problems arose concerning the input. Due to the sensors' nature/manufacturing, the collected data sometimes contained corrupted data. For this purpose, two (2) ways to handle the errors were developed.

(a) Correct Input Algorithm

The *Correct Input* algorithm was developed to fix the corrupted data that sometimes was produced by the sensors. As stated in Chapter 3 – section 3.6, each sensor collects data with an approximate rate of 15 angle readings per second and a step of 0.7°, meaning that each transition from one angle reading to the next differs at most 0.7°. Sometimes, due to the sensor's malfunction, the collected data contained "rubbish"/error values. The "rubbish" data can be translated as "jumps" in angle values of at least 2°-3° (way more than the allowed step of 0.7°). Moreover, these jumps occurred at least 2°-3° forwards or backwards, a fact that is not acceptable, since the angle readings only increase (therefore there is no decrease) at the passing of time.

The algorithm starts by traversing the “input_for_bearing.dat” before it is forwarded as input to the first stage of the *Bearing Definer*. The algorithm checks the angles of the file and how they change over time. If an angle value differs from the previous angle value more than 0.8° (0.1° more than the allowed step of 0.7°), we set the current angle value as (previous angle value + 0.7°). We repeat the procedure for “min_times” (as defined in the “parameters.txt”), ensuring that successor angles will appear at least “min_times” in the “input_for_bearing.dat” and that their values will not have a difference greater than 0.7°. In this way, the input is well organized and each transition in the angle values is gradually smooth.

(b) Undefined values

The second problem was that sometimes the data values produced by the sensor were “undefined”, resulting in program crashes. This problem was tackled in a simpler way. While reading the “input_for_bearing.dat” file in the above step, each time an “undefined” value was read, the line was rejected and the program resumed with the next line.

4.3.2 Triangulation Detector

The second program of the flow is the ***Triangulation Detector*** and is responsible for detecting all triangulation areas in our AOI.

The program consists of six (6) stages:

i. Reading/Storing the input
ii. Computation of bearing paths/Formation of quadrilaterals
iii. Determination of relationship between the quadrilaterals
iv. Reducing into triads
v. Removing duplicate values
vi. Printing the final output

4.3.2.1 Stage 1: Reading/Storing the input

The program accepts a specific input file called “input_for_triangulation.txt”, which is created in Node-RED and consists of the sensor’s bearing error – divergence (in angle degrees), a parameter called “max_angle_diff” and the information for each sensor. The “max_angle_diff” represents an angle in degrees and is used in the filtering process of the quadrilaterals (more information is described later on this chapter). The information of each sensor contains the sensor’s name, its coordinates (x, y) in degrees and the bearing angles – in degrees – for each sensor, as produced by the previous program, the ***Bearing Definer***.

The program’s input file has the following format:

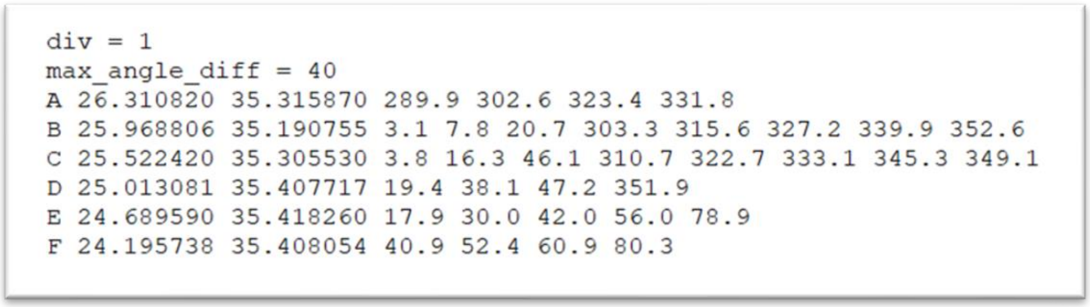
<Divergence>
<Max Angle Difference>
<Sensor’s Name> <Sensor’s Longitude> <Sensor’s Latitude> <Sensor’s Angles>

For example, given three (3) sensors:

Sensor A (long = 26.310820, lat = 35.315870), angles: 289.9, 302.6, 323.4, 331.8
Sensor B (long = 25.968806, lat = 35.190755), angles: 3.1, 7.8, 20.7, 303.3, 315.6, 327.2, 339.9, 352.6
Sensor C (long = 25.522420, lat = 35.305530), angles: 3.8, 16.3, 46.1, 310.7, 322.7, 333.1, 345.3, 349.1

a max angle difference of 40° and a divergence of 1°, the input file should have the following form:

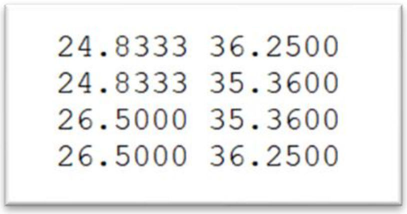
```
div = 1
max_angle_diff = 40
A 26.310820 35.315870 289.9 302.6 323.4 331.8
B 25.968806 35.190755 3.1 7.8 20.7 303.3 315.6 327.2 339.9 352.6
C 25.522420 35.305530 3.8 16.3 46.1 310.7 322.7 333.1 345.3 349.1
```



```
div = 1
max_angle_diff = 40
A 26.310820 35.315870 289.9 302.6 323.4 331.8
B 25.968806 35.190755 3.1 7.8 20.7 303.3 315.6 327.2 339.9 352.6
C 25.522420 35.305530 3.8 16.3 46.1 310.7 322.7 333.1 345.3 349.1
D 25.013081 35.407717 19.4 38.1 47.2 351.9
E 24.689590 35.418260 17.9 30.0 42.0 56.0 78.9
F 24.195738 35.408054 40.9 52.4 60.9 80.3
```

Figure 4.13. Input file "input_for_triangulation.txt"

Moreover, the program requires one more file from the same directory called "area_of_interest.txt". This file contains the coordinates of the four (4) vertices of the quadrilateral that represents our AOI. When attempting to determine the position of a transmitter, we take into consideration a specific area on the surface of the Earth the transmitter might be located in and apply the triangulation algorithm at the area described by the vertices of the respective quadrilateral. By following this procedure, we get rid of excessive computations, making our algorithm more efficient in terms of time and memory complexity.



```
24.8333 36.2500
24.8333 35.3600
26.5000 35.3600
26.5000 36.2500
```

Figure 4.14. File "area_of_interest.txt"

4.3.2.2 Stage 2: Computation of bearing paths/Formation of quadrilaterals

Upon reading all useful input information, we proceed to the next step, which is the creation of all quadrilaterals. As stated in Chapter 3 – section 3.5, in order to perform the triangulation method for two (2) or more transmitters, we need at least three (3) different sensors with their three

(3) different bearing angles. By taking into consideration each sensor's divergence, we have to create the required quadrilaterals per two (2), in order to be able to determine common areas between two quadrilaterals.

We begin by traversing the list of sensors – per two (2) – and computing the great-circle paths for each other, according to their bearing angles and the given divergence. At this point, we compute the absolute value of the difference between the two (2) bearing angles of the sensors. If the difference is less than the "max_angle_diff" parameter, we reject the current set of angles and proceed with the next one. This part of the algorithm is a modification that is analyzed in the ***General Notes*** section at the end of this section.

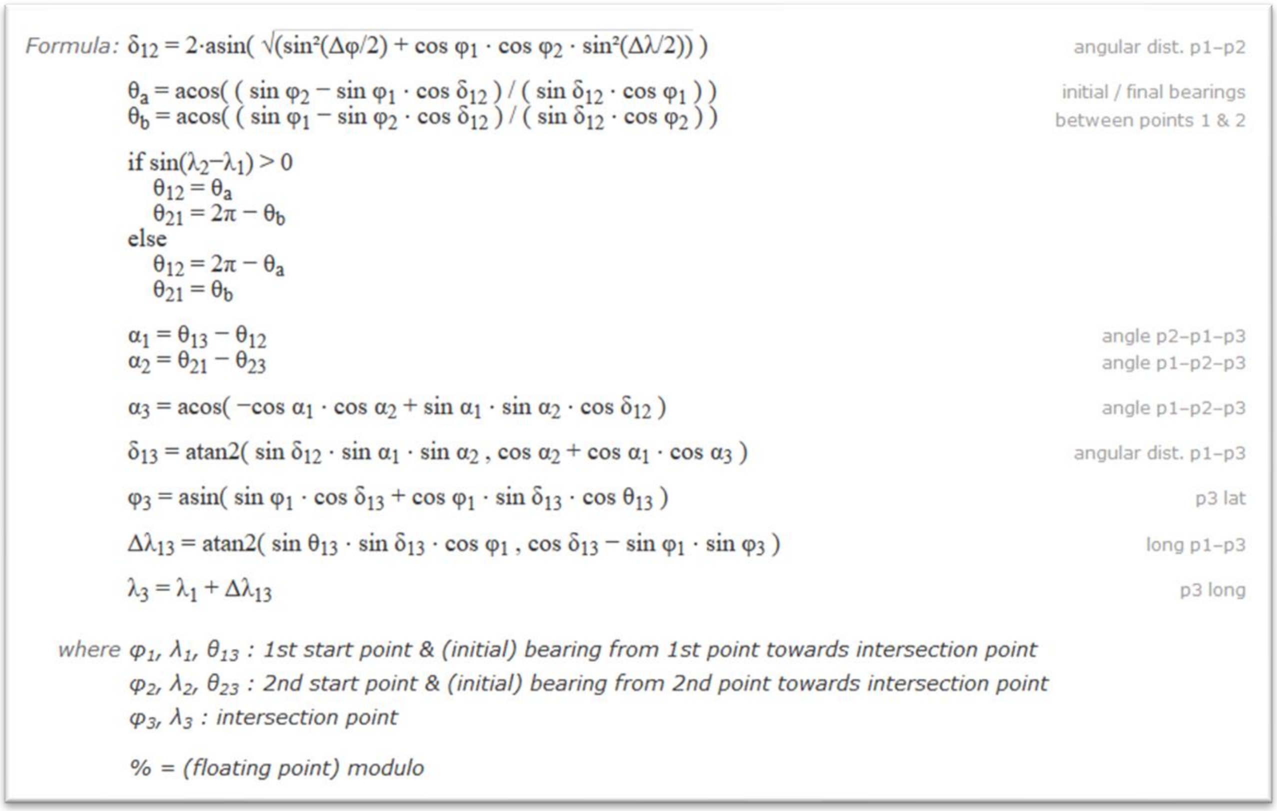
By applying the divergence formula (as described in Chapter 3 – section 3.4) on the bearing angles of each sensor, we end up with two (2) bearing paths deriving from the main bearing path of the corresponding sensor. For instance, for two (2) sensors A and B and their corresponding angles Aφ1 and Bφ1, the four (4) bearing paths that are formed are defined by the angles:

$$A\varphi 1 - div$$
$$A\varphi 1 + div$$
$$B\varphi 1 - div$$
$$B\varphi 1 + div$$

Next step will be to compute all intersection points between each of these paths, in order to form the quadrilaterals, as described in Chapter 3 – section 3.5.

The computation of the intersection points is performed based on the Intersection function provided by Google. The Intersection function is suitable for calculations on the basis of a spherical Earth (ignoring ellipsoidal effects), which is accurate enough for most purposes[2].

[2] In fact, the Earth is slightly ellipsoidal, so using a spherical model produces errors of up to 0.3%.



Formula: $\delta_{12} = 2\cdot\text{asin}(\ \sqrt{(\sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2))}\ )$ — angular dist. p1–p2

$\theta_a = \text{acos}(\ (\ \sin\varphi_2 - \sin\varphi_1 \cdot \cos\delta_{12}\ ) / (\ \sin\delta_{12} \cdot \cos\varphi_1\ )\ )$ — initial / final bearings

$\theta_b = \text{acos}(\ (\ \sin\varphi_1 - \sin\varphi_2 \cdot \cos\delta_{12}\ ) / (\ \sin\delta_{12} \cdot \cos\varphi_2\ )\ )$ — between points 1 & 2

```
if $\sin(\lambda_2 - \lambda_1) > 0$
    $\theta_{12} = \theta_a$
    $\theta_{21} = 2\pi - \theta_b$
else
    $\theta_{12} = 2\pi - \theta_a$
    $\theta_{21} = \theta_b$
```

$\alpha_1 = \theta_{13} - \theta_{12}$ — angle p2–p1–p3

$\alpha_2 = \theta_{21} - \theta_{23}$ — angle p1–p2–p3

$\alpha_3 = \text{acos}(\ -\cos\alpha_1 \cdot \cos\alpha_2 + \sin\alpha_1 \cdot \sin\alpha_2 \cdot \cos\delta_{12}\ )$ — angle p1–p2–p3

$\delta_{13} = \text{atan2}(\ \sin\delta_{12} \cdot \sin\alpha_1 \cdot \sin\alpha_2\ ,\ \cos\alpha_2 + \cos\alpha_1 \cdot \cos\alpha_3\ )$ — angular dist. p1–p3

$\varphi_3 = \text{asin}(\ \sin\varphi_1 \cdot \cos\delta_{13} + \cos\varphi_1 \cdot \sin\delta_{13} \cdot \cos\theta_{13}\ )$ — p3 lat

$\Delta\lambda_{13} = \text{atan2}(\ \sin\theta_{13} \cdot \sin\delta_{13} \cdot \cos\varphi_1\ ,\ \cos\delta_{13} - \sin\varphi_1 \cdot \sin\varphi_3\ )$ — long p1–p3

$\lambda_3 = \lambda_1 + \Delta\lambda_{13}$ — p3 long

where $\varphi_1, \lambda_1, \theta_{13}$ *: 1st start point & (initial) bearing from 1st point towards intersection point*

$\varphi_2, \lambda_2, \theta_{23}$ *: 2nd start point & (initial) bearing from 2nd point towards intersection point*

φ_3, λ_3 *: intersection point*

% = (floating point) modulo

Figure 4.15. Intersection formula [41]

If any intersection point is null (meaning two (2) of the bearing paths do not “meet” anywhere on the surface of the Earth), we ignore the specific bearing angles set and proceed with the next one. If all intersection points are not null, we check to see if at least one of them falls inside the area formed by the AOI’s quadrilateral. If no point meets the above condition, we ignore the specific bearing angles set and proceed with the next one. If the above condition is met – meaning that all intersection points are not null and at least one of them falls inside the AOI – we proceed with the creation of the respective quadrilateral. The quadrilateral’s vertices are the four (4) intersection points of the bearing paths and the name consists of the sensors’ names and the bearing angles from which the quadrilateral was created. We repeat the process for all bearing angles of all sensors – per two (2) – and store all created quadrilaterals in a list for future use. A quadrilateral, along with its vertices’ coordinates, is depicted below:

Quadrilateral: [A: 289.9, B: 3.1]

Vertices: (25.97, 35.40), (25.97, 35.41), (25.98, 35.41), (25.98, 35.40)

meaning that the above quadrilateral was formed by sensor A's bearing angle 289.9° and sensor B's bearing angle 3.1° and its vertices' coordinates are the intersection points of the four (4) bearing paths defined by these two (2) bearing angles.

Notes for Intersection Formula [41]

- ***Accuracy***: The Earth is generally ellipsoidal – more specifically oblate ellipsoidal – having an equatorial and polar radius of 6.378 km and 6.357 km respectively. The radius of curvature varies locally, ranging from 6.336 km (at the Equator) to 6.399 km (at the poles). The average radius of the Earth is currently accepted to be 6.371 km. Due to the Earth's shape, utilizing spherical geometry produces minor errors (since the Earth is not quite a sphere). When crossing the Equator, such errors may reach up to 0.55%, though generally, they are less than 0.3%, depending on latitude and the direction of travel.
- ***Bearings***: All bearings are measured with respect to true North (0°: N, 90°: E).
- ***Trigonometry functions***: Trigonometry functions accept arguments in radians; following this principle, longitude, latitude and angle bearings in degrees (either decimal or degrees/minutes/seconds) need to be converted to radians with the formula: rad=π*deg/180. When utilizing signed decimal degrees to convert radians back to degrees (deg=180*rad/π), West is negative. For bearings, values in the range [-π, +π] must be converted to [0, +2π]. This can be achieved by using the formula: (brng+2*π)%2*π, where % is the modulo operator.

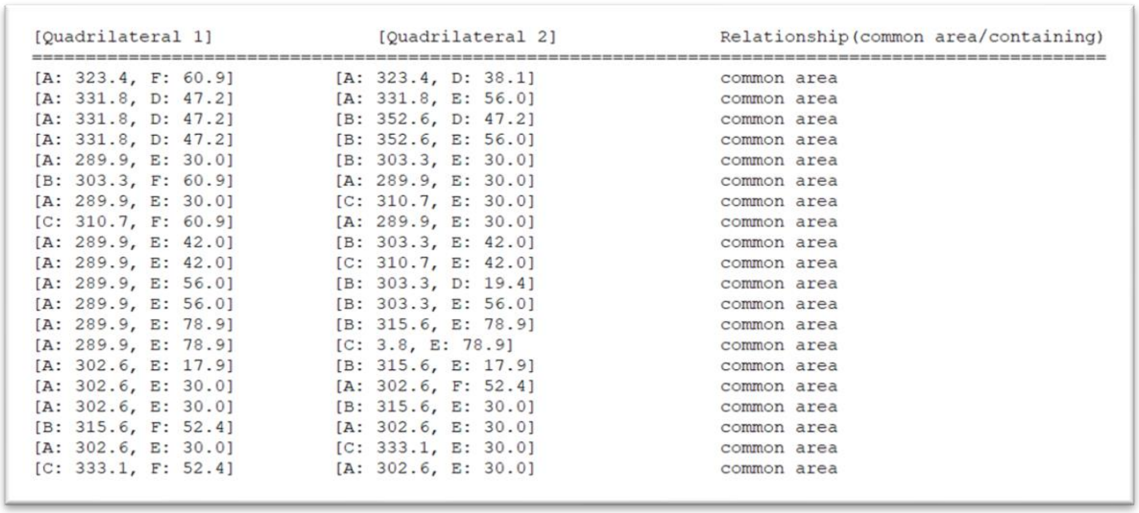
4.3.2.3 Stage 3: Determination of relationship between the quadrilaterals

After the formation of all possible acceptable quadrilaterals, we proceed to find if they have any kind of relationship with one another. Two (2) quadrilaterals may be related in three (3) ways:

i. Common area with each other
ii. "Containing" relationship (one quadrilateral contains another)
iii. No relationship at all

We begin by traversing the list of quadrilaterals – per two (2) – and search for any kind of relationship between them. If a common area or a containing relationship is detected, we store the quadrilaterals in a list, along with their relationship. If no relationship is detected, we ignore them and proceed with the next pair. At the end of this step, we have found all triangulation areas between the intersections of the bearing paths of all sensors in our list.

Upon determining all triangulation areas, we need to process the output, in order to be exported in the desired format for the next program, the ***Centroid Detector***. At this point, the output has the following format:

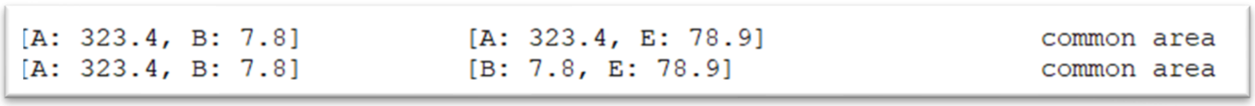


```
[Quadrilateral 1]           [Quadrilateral 2]           Relationship(common area/containing)
=================================================================================================
[A: 323.4, F: 60.9]         [A: 323.4, D: 38.1]         common area
[A: 331.8, D: 47.2]         [A: 331.8, E: 56.0]         common area
[A: 331.8, D: 47.2]         [B: 352.6, D: 47.2]         common area
[A: 331.8, D: 47.2]         [B: 352.6, E: 56.0]         common area
[A: 289.9, E: 30.0]         [B: 303.3, E: 30.0]         common area
[B: 303.3, F: 60.9]         [A: 289.9, E: 30.0]         common area
[A: 289.9, E: 30.0]         [C: 310.7, E: 30.0]         common area
[C: 310.7, F: 60.9]         [A: 289.9, E: 30.0]         common area
[A: 289.9, E: 42.0]         [B: 303.3, E: 42.0]         common area
[A: 289.9, E: 42.0]         [C: 310.7, E: 42.0]         common area
[A: 289.9, E: 56.0]         [B: 303.3, D: 19.4]         common area
[A: 289.9, E: 56.0]         [B: 303.3, E: 56.0]         common area
[A: 289.9, E: 78.9]         [B: 315.6, E: 78.9]         common area
[A: 289.9, E: 78.9]         [C: 3.8, E: 78.9]           common area
[A: 302.6, E: 17.9]         [B: 315.6, E: 17.9]         common area
[A: 302.6, E: 30.0]         [A: 302.6, F: 52.4]         common area
[A: 302.6, E: 30.0]         [B: 315.6, E: 30.0]         common area
[B: 315.6, F: 52.4]         [A: 302.6, E: 30.0]         common area
[A: 302.6, E: 30.0]         [C: 333.1, E: 30.0]         common area
[C: 333.1, F: 52.4]         [A: 302.6, E: 30.0]         common area
```

Figure 4.16. Output file with the quadrilaterals' relationships

4.3.2.4 Stage 4: Reducing into triads

The output consists of the information about the two (2) quadrilaterals (sensors and bearing angles), along with their relationships. Taking a closer look at the output, we can observe that some quadrilaterals share the same sensor and bearing angle. For instance:



```
[A: 323.4, B: 7.8]          [A: 323.4, E: 78.9]                common area
[A: 323.4, B: 7.8]          [B: 7.8, E: 78.9]                  common area
```

Figure 4.17. Before reduction into triads

we observe that quadrilaterals [A: 323.4, B: 7.8] and [A: 323.4, E: 78.9] share the same sensor A and angle 323.4°. This information can be reduced to a triad, consisting of the names of the three (3) sensors participating in the triangulation and their respective bearing angles. Following this principle, the above quadrilaterals can be reduced into the following triad:

(A: 323.4, B: 7.8, E: 78.9)

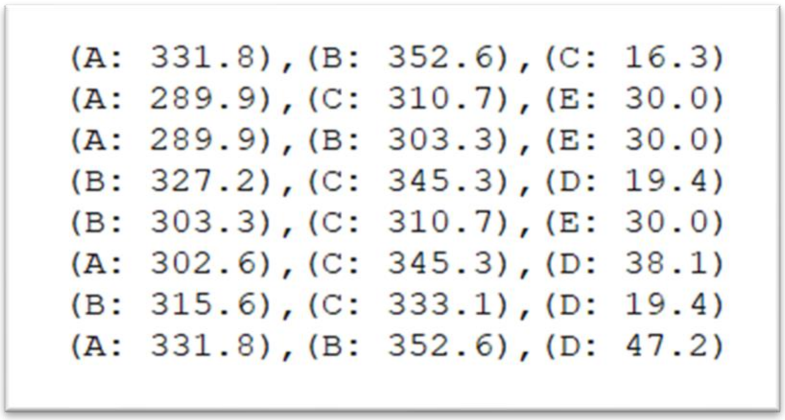
We perform this procedure to all elements of the output, until all elements have been reduced to their corresponding triads.

4.3.2.5 Stage 5: Removing duplicate values

Upon producing all triads, there is a possibility of having duplicate lines in our output. The next to last step of the ***Triangulation Detector*** is to remove all duplicate lines from the triads output. After the duplicates' removal, our program's output is ready to be printed.

4.3.2.6 Stage 6: Printing the final output

The final output of the ***Triangulation Detector*** is the "triads.txt" file. It contains all triangulation areas ("containing" relationship – "common area" between quadrilaterals reduced into triads), ready to be used as input to the final program of the current project, the ***Centroid Detector***, which will determine the exact position of a potential transmitter and project its position on the map.



```
(A: 331.8),(B: 352.6),(C: 16.3)
(A: 289.9),(C: 310.7),(E: 30.0)
(A: 289.9),(B: 303.3),(E: 30.0)
(B: 327.2),(C: 345.3),(D: 19.4)
(B: 303.3),(C: 310.7),(E: 30.0)
(A: 302.6),(C: 345.3),(D: 38.1)
(B: 315.6),(C: 333.1),(D: 19.4)
(A: 331.8),(B: 352.6),(D: 47.2)
```

Figure 4.18. Final output file "triads.txt"

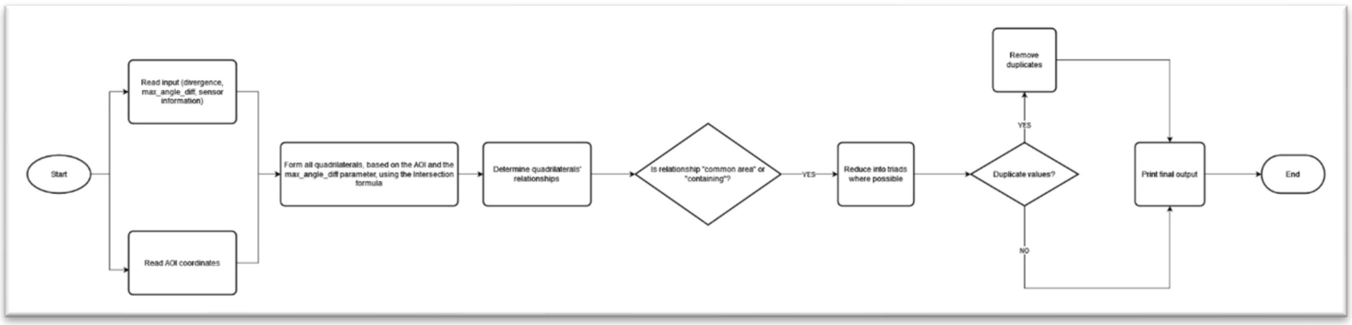

Start
Read input (divergence, max_angle_diff, sensor information)
Read AOI coordinates
Form all quadrilaterals, based on the AOI and the max_angle_diff parameter, using the intersection formula
Determine quadrilaterals' relationships
Is relationship "common area" or "containing"?
YES
Reduce into triads where possible
Duplicate values?
YES
Remove duplicates
NO
Print final output
End

Figure 4.19. *Triangulation Detector* flowchart

General Notes

The ***Triangulation Detector*** algorithm is an extension of my Bachelor's Thesis "***Determining triangulation areas from multiple sensors using specific bearing precision/accuracy***", with many modifications and bug fixes, in order to make the algorithm more efficient. The most important modification is the introduction of the AOI quadrilateral. In the earlier versions of the algorithm, all possible quadrilaterals were computed and stored, resulting in unnecessary use of space and memory. No quadrilaterals were rejected and the number of comparisons between them, in order to find all triangulation areas, was huge, as the earlier versions were based on the “greedy” approach. With the introduction of the AOI quadrilateral, the number of computed quadrilaterals and thus, triangulation areas, is dramatically reduced, because we are now keeping only the points that are of great interest to us and reject all points that fall a great distance away from our AOI.

Another important modification was the introduction of the “max_angle_diff” parameter. The “max_angle_diff” helps reduce the number of useless triangulations. While traversing the list of sensors – per two (2) – we compute the absolute value of the difference between the two (2) bearing angles of the sensors. If the value is less than the “max_angle_diff”, we reject the current pair of angles and proceed with the next one. For instance, for a sensor A with a bearing angle of 45° and a sensor B with a bearing angle of 350°, the difference between their bearing angles is equal to 55°.

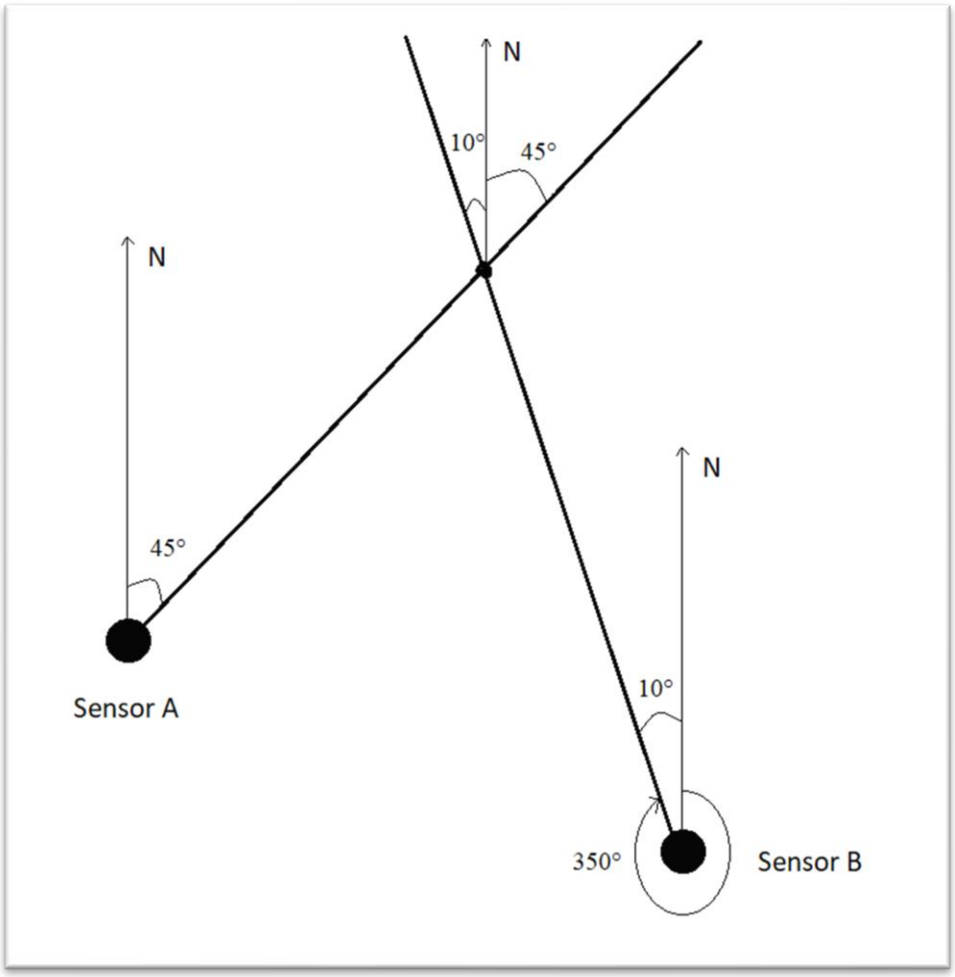


Figure 4.20. Interior-Exterior angles formed between parallel norths

This makes use of the properties of interior-exterior angles, formed between parallel lines. In our case, the parallel lines are the paths corresponding to the true North (marked with **N** in Figure 4.20). This process is approximate: in reality, paths corresponding to the true North are not parallel – due to the Earth's curvature – but rather meet at the Earth's poles. In our case, though, for a small AOI of some square kilometers on the surface of the Earth (plus, for simplicity of computation), it is safe to accept that these paths are parallel to each other and therefore, we can take advantage of the interior-exterior angles' properties.

By rejecting the pair of bearing angles whose absolute difference is less than the "max_angle_diff", we set a "floor" threshold for two (2) angles to have at least a specific difference between each other. If that difference is too small, this translates to the two (2) bearing angles being too close to each other and thus, being approximately almost the same; therefore, there is no point in computing a quadrilateral whose vertices derive from the same bearing angles.

Finally, we fixed a major bug that occurred when reducing two (2) quadrilaterals in a triad. In some cases, two (2) quadrilaterals with common area were created from the same sensor (e.g., sensor A) but with a different bearing angle (e.g., [A: 45.0, B: 46.0] and [A: 19.0, E: 339.0]). This type of relationship is not acceptable, as we are only interested in the common area of quadrilaterals from different sensors and therefore, has to be rejected. We handled this case by ignoring the specific pair of quadrilaterals.

4.3.3 Centroid Detector

After determining the triangulation areas using the ***Triangulation Detector***, the final step of the project is to estimate the transmitter's position and project it on the map. The output from section 4.3.2 gives us information about a transmitter's potential position. However, this position is approximate. According to this information, the transmitter could be anywhere inside the triangulation areas. Therefore, a more accurate way of determining its position is needed. For this purpose, the computation of the triangulation areas' centroid is the focus of our final program. Since the output from the *Triangulation Detector* consists of triads, each one containing three (3) sensors with three (3) of their bearing angles, the intersections of the bearing paths between the three (3) sensors give three (3) vertices of a triangle. The computation of the triangle's centroid gives a more accurate representation of the transmitters' potential positions.

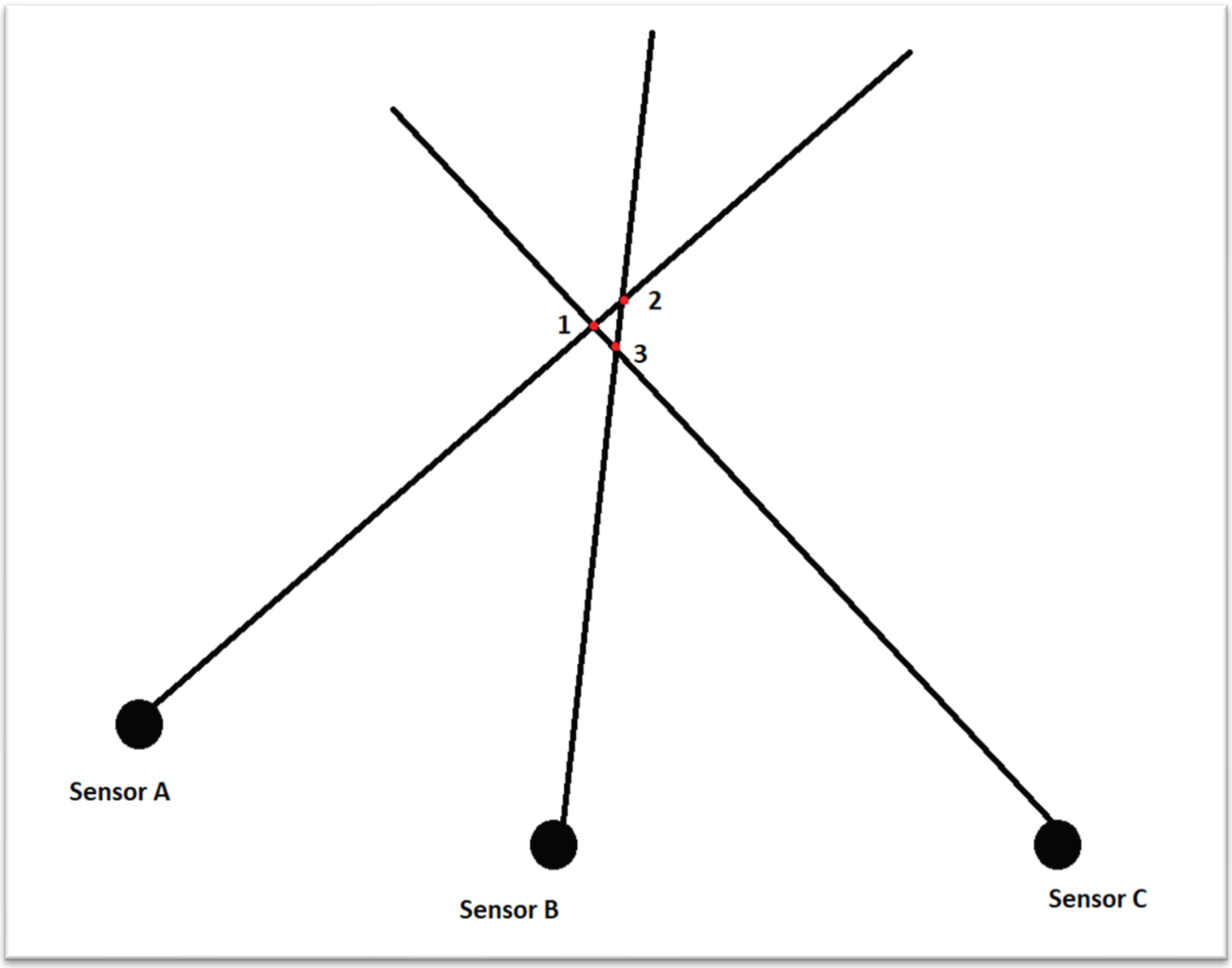


Figure 4.21. Triangle formation by the bearing paths of three (3) sensors

The program's input files are both the "input_for_triangulation.txt" file from the previous program, since it contains all information about the sensors (their coordinates and bearing angles), and the "triads.txt" file that is the final output of the previous program. The "triads.txt" contains all triangulation areas, so along with the sensor information, we are able to compute the triangle centroids.

Upon reading the two (2) input files, our aim is to create each triangle that corresponds to each triad in the “triads.txt” file and compute its centroid. Since we have all the information we need about the sensors, we can use the ***Intersection formula***, as described in Figure 4.12, to find the three (3) vertices of each triangle. The triangles’ vertices are the intersections of the bearing paths (defined by the angle bearings) that participate in the triads. For instance, for the triad:

(A: 302.6), (D: 19.4), (E: 42.0)

the vertices of the respective triangle are the intersection points of the bearing paths between:

A and D (intersection point 1)
A and E (intersection point 2)
D and E (intersection point 3)

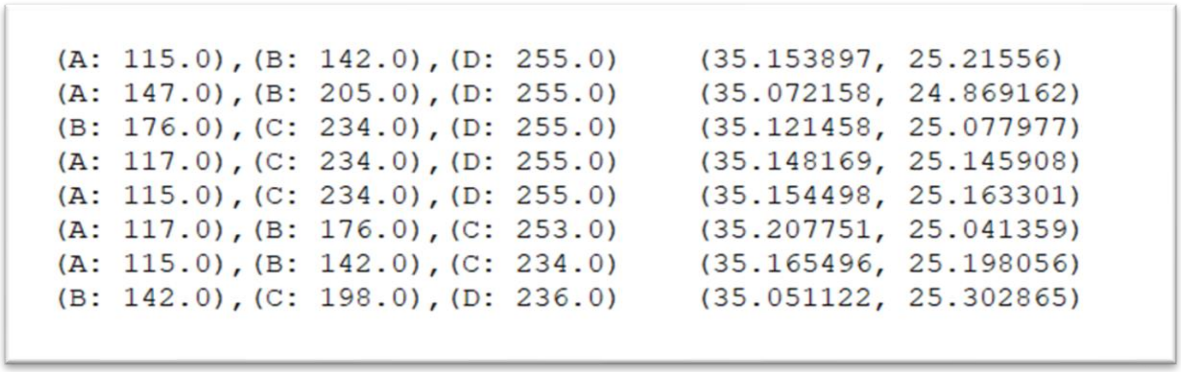
By acquiring the vertices for each triangle, we can then compute its centroid using the formula:

$$\mathbf{G\ (x, y) = ((x1+x2+x3)/3,\ (y1+y2+y3)/3) \qquad (III)}$$

where:

(x1, y1): coordinates of intersection point 1
(x2, y2): coordinates of intersection point 2
(x3, y3): coordinates of intersection point 3

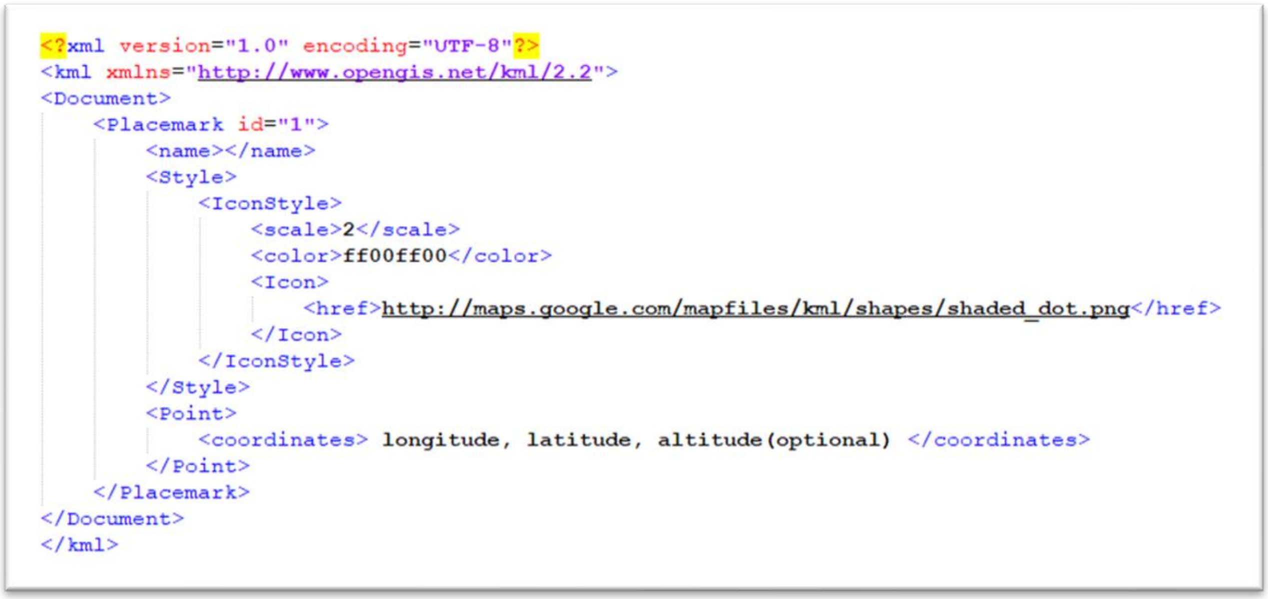
The formula suggests that the coordinates G (x, y) of a triangle’s centroid are defined by the average of the coordinates of its three (3) vertices. We repeat the procedure for all the triads in the “triads.txt” file and save all centroids in a list for the final print. The centroids are the first half of the final output of our program, since they are the coordinates of the transmitter, and are printed in the file “centroid_coords.txt”, along with the triad they correspond to.



```
(A: 115.0),(B: 142.0),(D: 255.0)    (35.153897, 25.21556)
(A: 147.0),(B: 205.0),(D: 255.0)    (35.072158, 24.869162)
(B: 176.0),(C: 234.0),(D: 255.0)    (35.121458, 25.077977)
(A: 117.0),(C: 234.0),(D: 255.0)    (35.148169, 25.145908)
(A: 115.0),(C: 234.0),(D: 255.0)    (35.154498, 25.163301)
(A: 117.0),(B: 176.0),(C: 253.0)    (35.207751, 25.041359)
(A: 115.0),(B: 142.0),(C: 234.0)    (35.165496, 25.198056)
(B: 142.0),(C: 198.0),(D: 236.0)    (35.051122, 25.302865)
```

Figure 4.22. Centroid Detector's final output "centroid_coords.txt"

The second step to complete the program is to project the transmitter's position on the map. For this purpose, a KML (Keyhole Markup Language) file called "centroidKML.kml" is produced, which contains all centroid points with a colored icon, in order to be easily distinguishable on the map. The KML's format is:



```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
    <Placemark id="1">
        <name></name>
        <Style>
            <IconStyle>
                <scale>2</scale>
                <color>ff00ff00</color>
                <Icon>
                    <href>http://maps.google.com/mapfiles/kml/shapes/shaded_dot.png</href>
                </Icon>
            </IconStyle>
        </Style>
        <Point>
            <coordinates> longitude, latitude, altitude(optional) </coordinates>
        </Point>
    </Placemark>
</Document>
</kml>
```

Figure 4.23. KML file format

where longitude, latitude are the centroid’s coordinates and altitude is an optional parameter representing the altitude from the Earth’s surface (for simplicity reasons, we set the altitude parameter to zero (0)).

Every time the three (3) programs are executed (according to the “run_times” parameter – Section 3.3.1), a different KML file is produced, containing all centroid points of the respective “triads.txt” files. The “centroidKML.kml” is located in a directory, from which it is read by Google Earth Pro. Google Earth Pro offers the ability to upload a KML file and project it on the Earth’s surface. By setting Google Earth Pro to read the “centroidKML.kml” file from the directory it is located in, we are able to project our results on the Earth’s surface in real time and thus, our program’s final output is officially completed. In Figure 4.24, a screenshot of Google Earth Pro is provided:

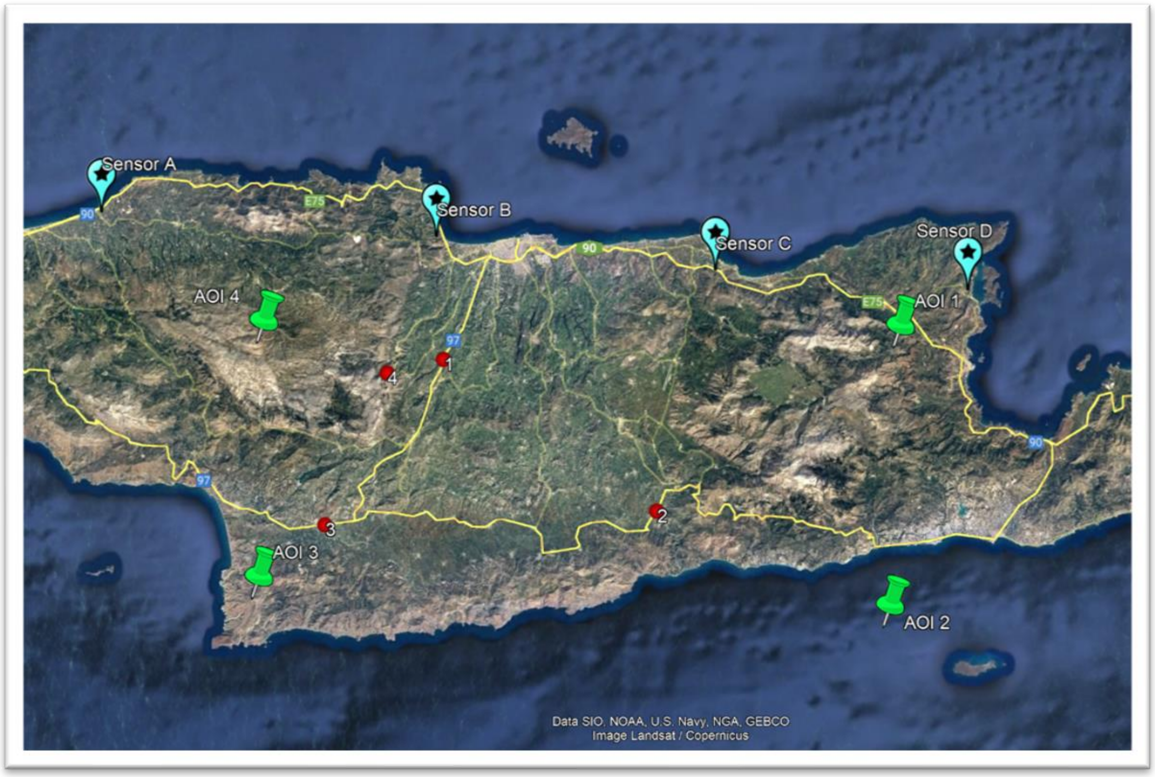

Sensor A
Sensor B
Sensor C
Sensor D
AOI 4
AOI 1
AOI 3
AOI 2
1
2
3
4
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
Image Landsat / Copernicus

Figure 4.24. Transmitters' position projection on Google Earth Pro

where the transmitters are marked with red dots and our AOI with green pins.

Last but not least, the *Centroid Detector*'s flowchart is presented below:

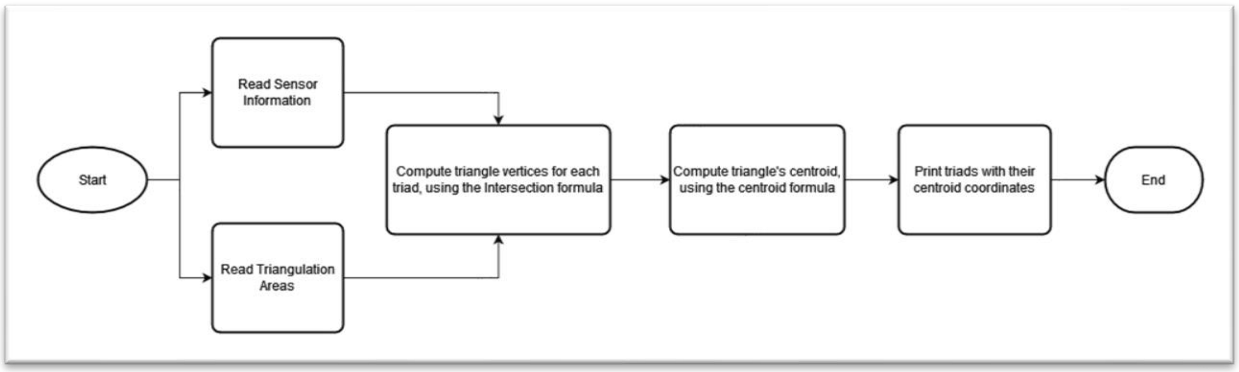

Start
Read Sensor Information
Read Triangulation Areas
Compute triangle vertices for each triad, using the Intersection formula
Compute triangle's centroid, using the centroid formula
Print triads with their centroid coordinates
End

Figure 4.25. *Centroid Detector* flowchart

Finally, the final flowchart displaying all stages of the project is presented below:

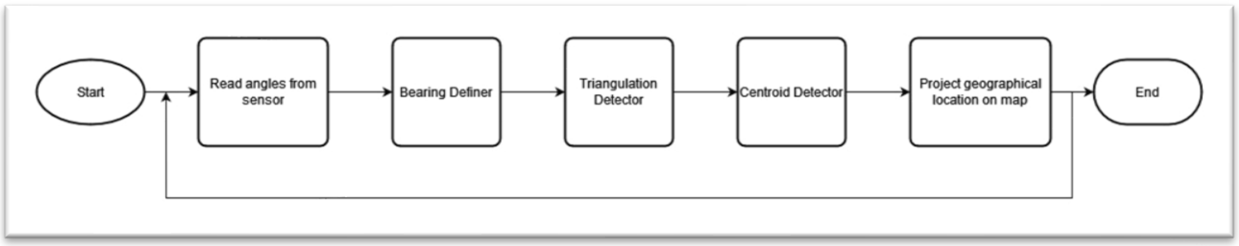


Figure 4.26. Final Flowchart

4.4 Experiment / Results

4.4.1 Experiment

In order to test the program’s validity and efficiency, four (4) sensors were deployed on coastal areas of northern Crete, from Rethymno district to Lasithi district. Our area of interest (AOI) laid on central Crete, as depicted in Figure 4.27.

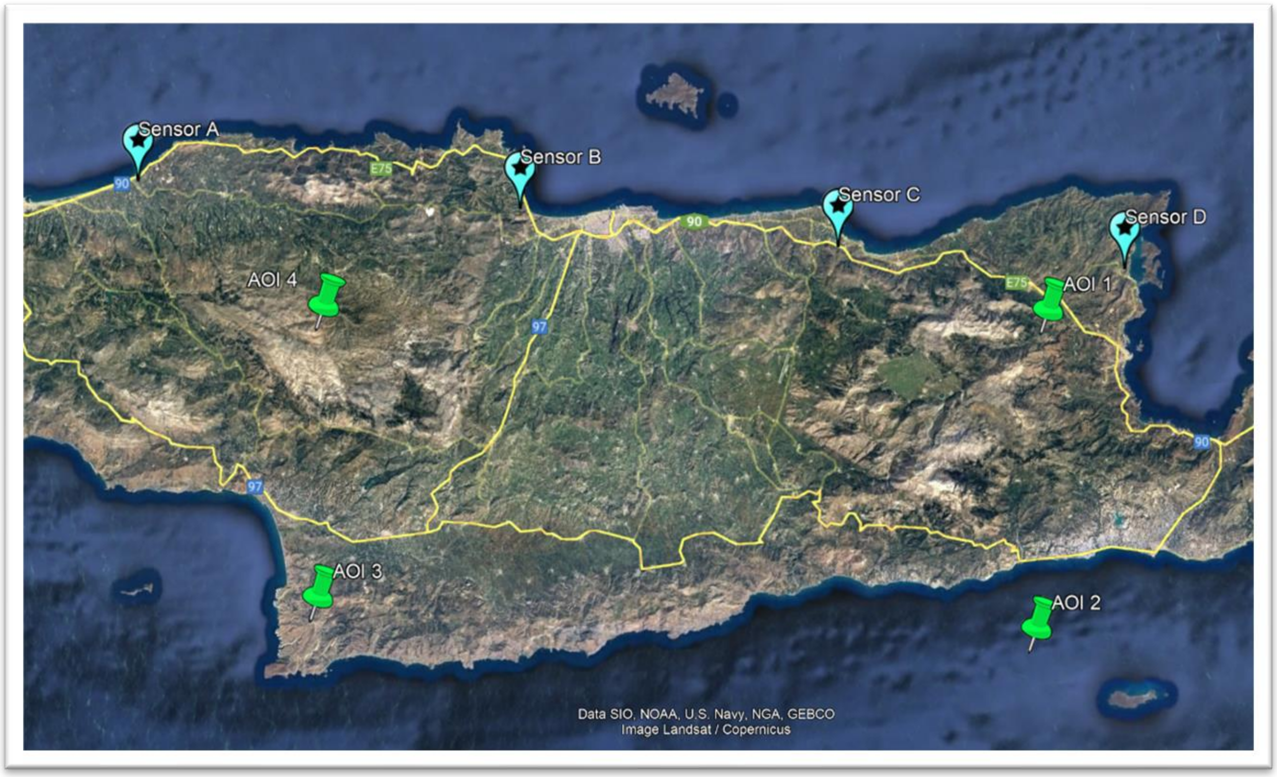

Sensor A
Sensor B
Sensor C
Sensor D
AOI 1
AOI 2
AOI 3
AOI 4
90
97
E75
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
Image Landsat / Copernicus

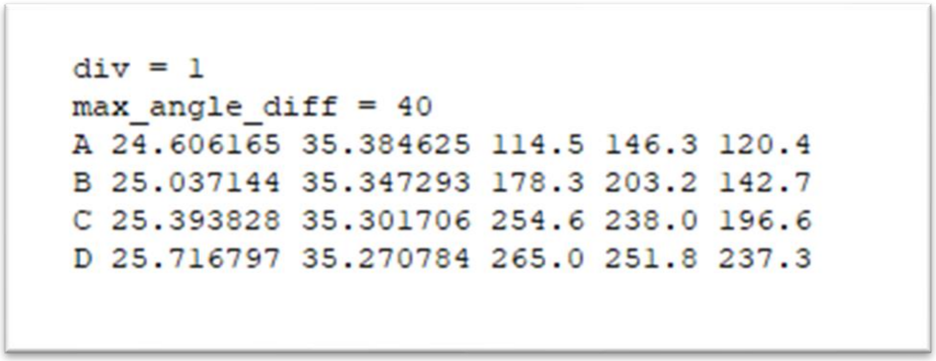
Figure 4.27. Our experiment’s environment

A number of transmitters performed an activity inside our AOI. The aim of the experiment was to estimate the transmitters’ positions in the AOI.

We deployed and ran the program twice. The first time was to detect all triangulation areas and possible locations of transmitters in our AOI, whereas the second time was to validate the results. In case of a false detection the first time, the second time would omit potential bias.

4.4.2 Results

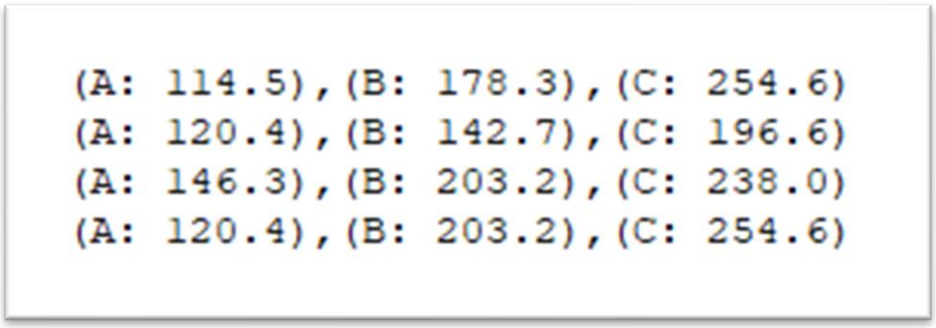
At first run, the input file that was created by Node-RED in order to be forwarded in the *Triangulation Detector* algorithm was the following:



```
div = 1
max_angle_diff = 40
A 24.606165 35.384625 114.5 146.3 120.4
B 25.037144 35.347293 178.3 203.2 142.7
C 25.393828 35.301706 254.6 238.0 196.6
D 25.716797 35.270784 265.0 251.8 237.3
```

Figure 4.28. First run “input_for_triangulation.txt”

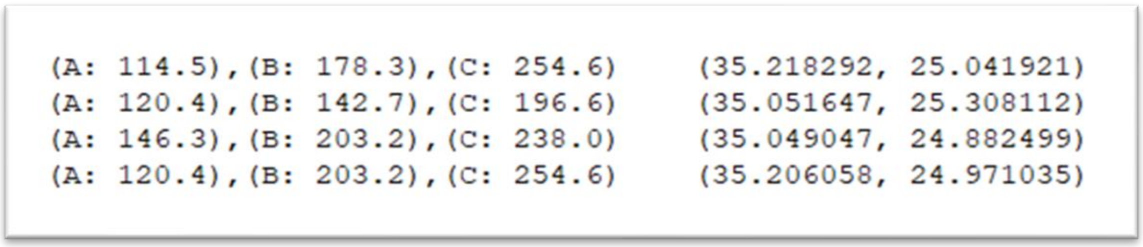
where the sensors’ bearing angles were computed by an instance of the *Bearing Definer* for every sensor. The *Triangulation Detector*, using the above input, produced the following triads/triangulation areas:



```
(A: 114.5),(B: 178.3),(C: 254.6)
(A: 120.4),(B: 142.7),(C: 196.6)
(A: 146.3),(B: 203.2),(C: 238.0)
(A: 120.4),(B: 203.2),(C: 254.6)
```

Figure 4.29. First run “triads.txt”

Similarly, forwarding the “triads.txt” file to the *Centroid Detector* program, the final “centroid_coords.txt” is produced:



```
(A: 114.5),(B: 178.3),(C: 254.6)    (35.218292, 25.041921)
(A: 120.4),(B: 142.7),(C: 196.6)    (35.051647, 25.308112)
(A: 146.3),(B: 203.2),(C: 238.0)    (35.049047, 24.882499)
(A: 120.4),(B: 203.2),(C: 254.6)    (35.206058, 24.971035)
```

Figure 4.30. First run “centroid_coords.txt”

A screenshot from Google Earth Pro is presented below:

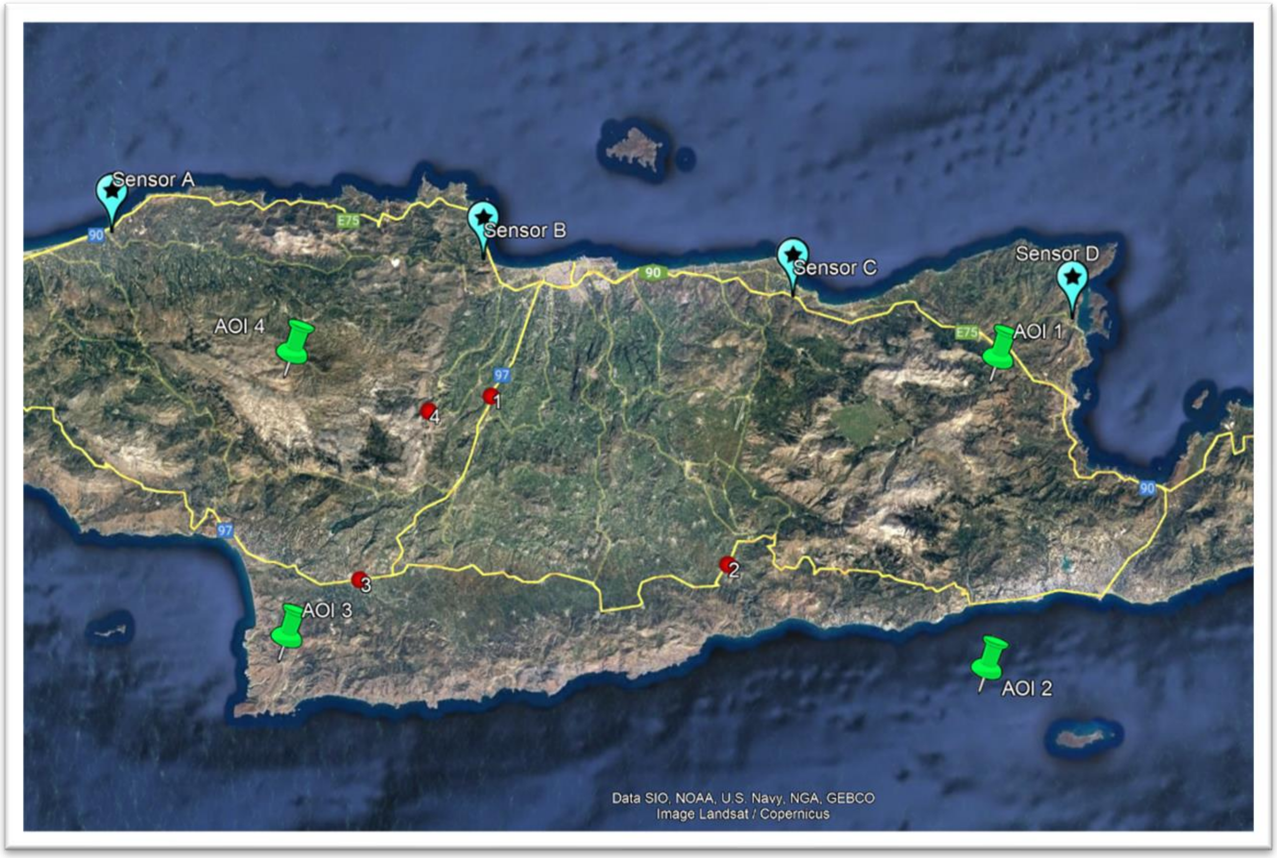
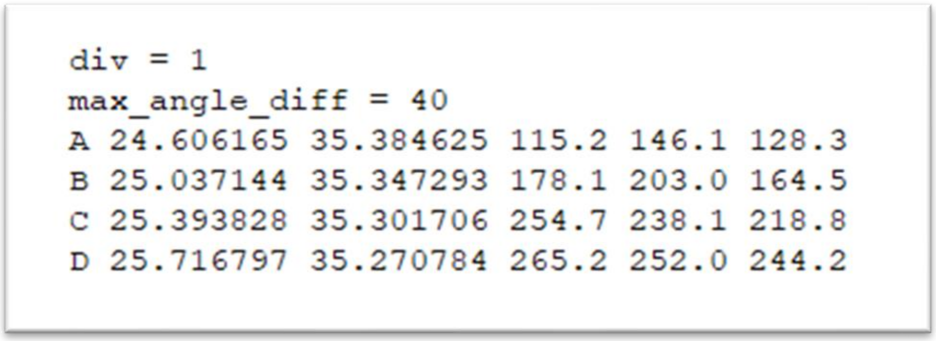

Sensor A
Sensor B
Sensor C
Sensor D
AOI 4
AOI 1
AOI 3
AOI 2
1
2
3
4
90
E75
97
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
Image Landsat / Copernicus

Figure 4.31. First run transmitters’ position estimation

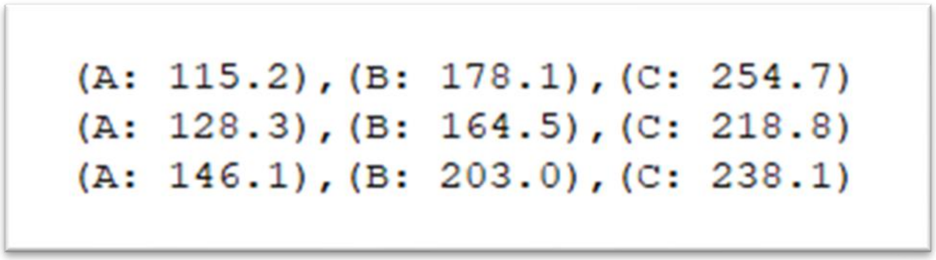
The second run of the program was performed to validate previous results and remove any false detections. Similarly, after the execution of the *Bearing Definer* for each sensor, the “input_for_triangulation.txt” was formed:



```
div = 1
max_angle_diff = 40
A 24.606165 35.384625 115.2 146.1 128.3
B 25.037144 35.347293 178.1 203.0 164.5
C 25.393828 35.301706 254.7 238.1 218.8
D 25.716797 35.270784 265.2 252.0 244.2
```

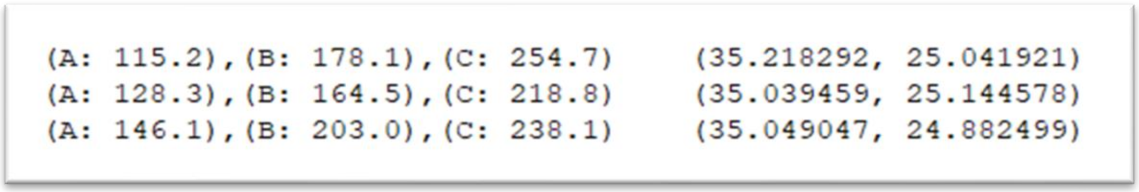
Figure 4.32. Second run “input.txt”

The second run’s triads/triangulation areas and final “centroid_coords.txt” are presented below:



```
(A: 115.2),(B: 178.1),(C: 254.7)
(A: 128.3),(B: 164.5),(C: 218.8)
(A: 146.1),(B: 203.0),(C: 238.1)
```

Figure 4.33. Second run “triads.txt”



```
(A: 115.2),(B: 178.1),(C: 254.7)    (35.218292, 25.041921)
(A: 128.3),(B: 164.5),(C: 218.8)    (35.039459, 25.144578)
(A: 146.1),(B: 203.0),(C: 238.1)    (35.049047, 24.882499)
```

Figure 4.34. Second run "centroid_coords.txt”

The final screenshot from Google Earth Pro presents now three (3) geographical positions, instead of four (4):

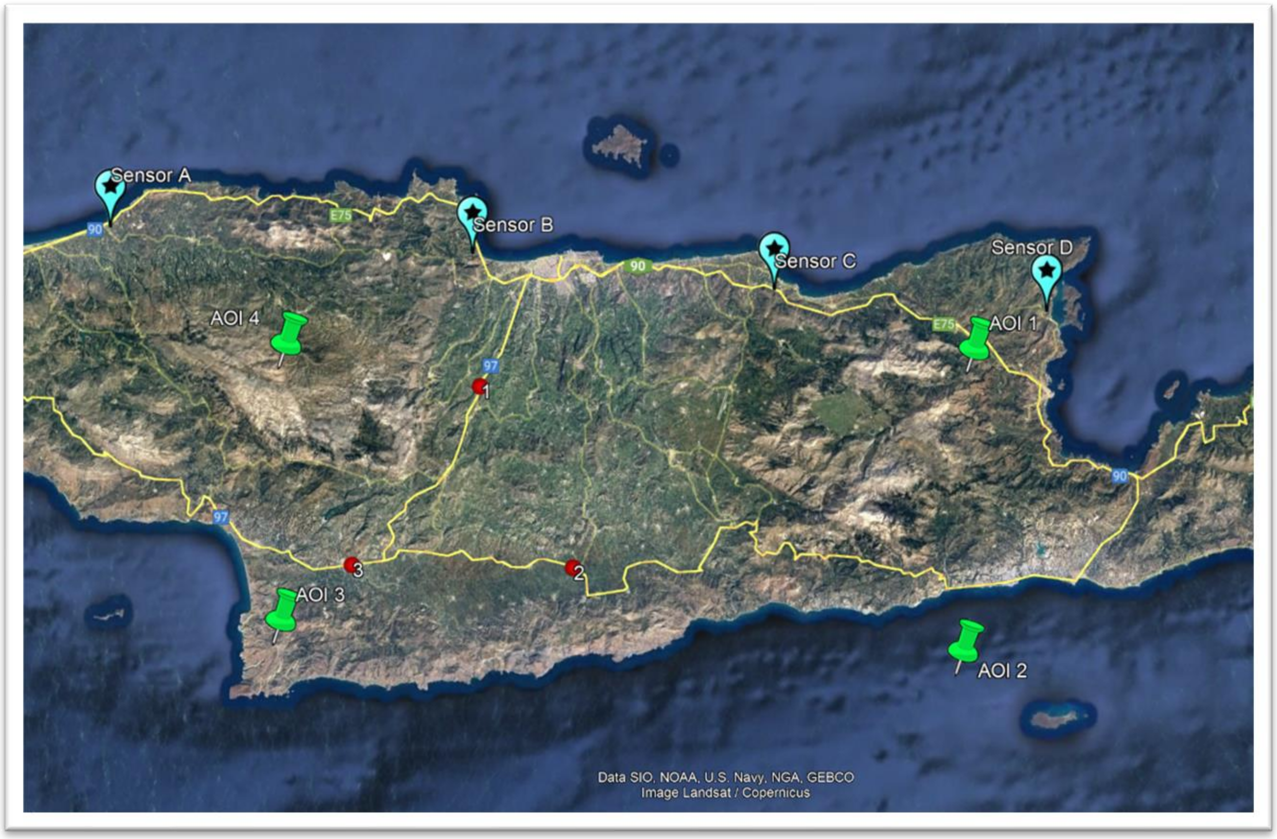

Sensor A
Sensor B
Sensor C
Sensor D
AOI 4
AOI 1
AOI 3
AOI 2
1
2
3
E75
90
97
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
Image Landsat / Copernicus

Figure 4.35. Second run transmitters' position estimation

By observing the screenshot, we can conclude that one (1) out of the four (4) positions in the first run was a false detection, since it does not exist in the second run. Moreover, we observe that transmitter #2 has changed position and has a direction towards West, according to its initial position in Figure 4.31 and its final position in Figure 4.35. The other two (2) transmitters appear to be in a fixed position.

Chapter 5 - Conclusion / Future Work

5.1 Conclusions

In the last years, the topic of localization has been of prime research interest, where the need for accurate position estimation has grown massively. The increasing number of sensor network applications demands accurate detection rates in applications like GPS positioning, mobile phone technologies, robotics etc. One of the localization techniques – and the focus of the current thesis – is the process of triangulation.

In a Fixed Sensor Network (FSN), where sensors are connected wirelessly via 4G, the need to estimate a transmitter's position entering our area of interest (AOI) in real time is of prime importance. The sensors collect large amounts of data over a large area of several square kilometers on the Earth's surface. The collected data needs analysis and processing, in order to find high-intensity bearings and therefore, be able to estimate the transmitter's position. The data is then used in triangulation algorithms, in order to find all possible triangulation areas the transmitter may exist in and finally, using the triangulation areas, the transmitter's position is estimated and projected on a map in real time.

In the current thesis we presented three (3) different algorithms for data processing and position estimation. The ***Bearing Definer*** is responsible for the analysis of the data collected by the sensors, as it detects and exports all high-intensity bearings received by the sensors. The ***Triangulation Detector*** receives the high-intensity bearings collected by the previous stage and performs triangulation algorithms, in order to find the triangulation areas in which the transmitter may be located. Finally, the ***Centroid Detector*** receives the areas produced by the previous stage, computes the exact geographical position (longitude, latitude) of the transmitter and projects the position in Google Maps Pro.

5.2 Future Work

A very important addition to the current program will be the insertion of a database, in order to store the sensors' data (name, coordinates, bearing angles) and other information that may prove important in the future. The use of a database will help reduce the number of ".txt" files in a single execution of the program, since all necessary information will be directly accessible through the database.

Moreover, in order to determine the “winning” bearing angles more efficiently in the first program of the flow, we plan on developing a Neural Network that will replace the ***Bearing Definer***’s job by making greater use of the respective weights in order to perform a better and more accurate decision.

References

[1] M. Sfendourakis, R. Nilavalan, and E. Antonidakis, “Sensors Networks - Localization and Topology,” in *2016 Third International Conference on Mathematics and Computers in Sciences and in Industry (MCSI)*, IEEE, Aug. 2016, pp. 143–154. doi: 10.1109/MCSI.2016.036.

[2] M. Gast and O. ’ Reilly, “802.11® Wireless Networks: The Definitive Guide,” 2002.

[3] S. Sharma, D. Kumar, and K. Kishore, “Wireless Sensor Networks-A Review on Topologies and Node Architecture,” *International Journal of Computer Science International Journal of Computer Science International Journal of Computer Science International Journal of Computer*, no. 2, pp. 19–25, 2013, [Online]. Available: www.ijcaonline.org

[4] Y. N. Singh, N. K. Verma, and R. K. Tripathi, “Two-tiered wireless sensor networks – base station optimal positioning case study,” *IET Wireless Sensor Systems*, vol. 2, no. 4, pp. 351–360, Dec. 2012, doi: 10.1049/iet-wss.2011.0152.

[5] Y. K. Joshi and M. Younis, “Autonomous recovery from multi-node failure in Wireless Sensor Network,” in *2012 IEEE Global Communications Conference (GLOBECOM)*, IEEE, Dec. 2012, pp. 652–657. doi: 10.1109/GLOCOM.2012.6503187.

[6] B. Sarwar, I. Bajwa, S. Ramzan, B. Ramzan, and M. Kausar, “Design and Application of Fuzzy Logic Based Fire Monitoring and Warning Systems for Smart Buildings,” *Symmetry (Basel)*, vol. 10, no. 11, p. 615, Nov. 2018, doi: 10.3390/sym10110615.

[7] M. Toloueiashtian and H. Motameni, “A new clustering approach in wireless sensor networks using fuzzy system,” *J Supercomput*, vol. 74, no. 2, pp. 717–737, Feb. 2018, doi: 10.1007/s11227-017-2153-0.

[8] G. Wu and C. Wu, “Research and application of node fuzzy identification and localization in wireless sensor networks,” *International Journal of Communication Systems*, vol. 34, no. 10, Jul. 2021, doi: 10.1002/dac.4835.

[9] B. Abood, A. Hussien, Y. Li, and D. Wang, “Energy Efficient Clustering in Wireless SensorNetworks Using Fuzzy Approach to Improve LEACH Protocol,” *INTERNATIONAL JOURNAL OF MANAGEMENT & INFORMATION TECHNOLOGY*, vol. 11, no. 2, pp. 2641–2656, Jun. 2016, doi: 10.24297/ijmit.v11i2.4856.

[10] M. Maksimović, V. Vujović, and V. Milošević, “Fuzzy logic and Wireless Sensor Networks – A survey,” *Journal of Intelligent & Fuzzy Systems*, vol. 27, no. 2, pp. 877–890, 2014, doi: 10.3233/IFS-131046.

[11] K. Kapitanova, S. H. Son, and K.-D. Kang, “Using fuzzy logic for robust event detection in wireless sensor networks,” *Ad Hoc Networks*, vol. 10, no. 4, pp. 709–722, Jun. 2012, doi: 10.1016/j.adhoc.2011.06.008.

[12] Kaur Arshdeep and Kaur Amrit, “Comparison of Mamdani-Type and Sugeno-Type Fuzzy Inference Systems for Air Conditioning System,” 2012.

[13] S. Garcia-Jimenez, A. Jurio, M. Pagola, L. De Miguel, E. Barrenechea, and H. Bustince, “Forest fire detection: A fuzzy system approach based on overlap indices,” *Appl Soft Comput*, vol. 52, pp. 834–842, Mar. 2017, doi: 10.1016/j.asoc.2016.09.041.

[14] M. Maksimovic, V. Vujovic, B. Perisic, and V. Milosevic, “Developing a fuzzy logic based system for monitoring and early detection of residential fire based on thermistor sensors,” *Computer Science and Information Systems*, vol. 12, no. 1, pp. 63–89, 2015, doi: 10.2298/CSIS140330090M.

[15] D. M. Majeed, H. W. Rabee, and Z. Ma, “Improving Energy Consumption Using Fuzzy-GA Clustering and ACO Routing in WSN,” in *2020 3rd International Conference on Artificial Intelligence and Big Data (ICAIBD)*, IEEE, May 2020, pp. 293–298. doi: 10.1109/ICAIBD49809.2020.9137446.

[16] A. Howard, M. J. Matarić, and G. S. Sukhatme, “Mobile Sensor Network Deployment using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem,” in *Distributed Autonomous Robotic Systems 5*, Tokyo: Springer Japan, 2002, pp. 299–308. doi: 10.1007/978-4-431-65941-9_30.

[17] E. Shakshuki, A. A. Elkhail, I. Nemer, M. Adam, and T. Sheltami, “Comparative Study on Range Free Localization Algorithms,” *Procedia Comput Sci*, vol. 151, pp. 501–510, 2019, doi: 10.1016/j.procs.2019.04.068.

[18] L. Doherty, K. S. J. pister, and L. El Ghaoui, “Convex position estimation in wireless sensor networks,” in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, IEEE, 2001, pp. 1655–1663. doi: 10.1109/INFCOM.2001.916662.

[19] A. Savvides, C.-C. Han, and M. B. Strivastava, “Dynamic fine-grained localization in Ad-Hoc networks of sensors,” in *Proceedings of the 7th annual international conference on Mobile computing and networking*, New York, NY, USA: ACM, Jul. 2001, pp. 166–179. doi: 10.1145/381677.381693.

[20] A. Nasipuri and K. Li, “A directionality based location discovery scheme for wireless sensor networks,” in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, New York, NY, USA: ACM, Sep. 2002, pp. 105–111. doi: 10.1145/570738.570754.

[21] N. Patwari, A. O. Hero, M. Perkins, N. S. Correal, and R. J. O’Dea, “Relative location estimation in wireless sensor networks,” *IEEE Transactions on Signal Processing*, vol. 51, no. 8, pp. 2137–2148, Aug. 2003, doi: 10.1109/TSP.2003.814469.

[22] D. Niculescu and Badri Nath, “Ad hoc positioning system (APS) using AOA,” in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, IEEE, 2003, pp. 1734–1743. doi: 10.1109/INFCOM.2003.1209196.

[23] R. Kaur and J. Malhotra, “Range Free Localization Techniques for Randomly Deployed WSN-A Survey,” *International Journal of Grid and Distributed Computing*, vol. 8, no. 6, pp. 57–66, Dec. 2015, doi: 10.14257/ijgdc.2015.8.6.07.

[24] V. M. Yuvashree AK, “Survey on Range Free-Localization Schemes Based on Anchors Count in WSN Model,” *IJIRST-International Journal for Innovative Research in Science & Technology*|, vol. 1, 2015, [Online]. Available: www.ijirst.org

[25] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. Abdelzaher, “Range-free localization schemes for large scale sensor networks,” in *Proceedings of the 9th annual international conference on Mobile computing and networking*, New York, NY, USA: ACM, Sep. 2003, pp. 81–95. doi: 10.1145/938985.938995.

[26] M. Sfendourakis, M. Zakynthinaki, E. Vasilaki, E. Antonidakis, and R. Nilavalan, “Coverage Area of a Localization Fixed Sensors Network System with the process of Triangulation,” *WSEAS TRANSACTIONS ON INFORMATION SCIENCE AND APPLICATIONS*, vol. 18, pp. 39–56, Jun. 2021, doi: 10.37394/23209.2021.18.7.

[27] M. Sfendourakis, R. Nilavalan, and E. Antonidakis, “Triangulation positioning system network,” *MATEC Web of Conferences*, vol. 125, p. 02069, Oct. 2017, doi: 10.1051/matecconf/201712502069.

[28] M. Sfendourakis *et al.*, “Area Coverage improvement of a Fixed Sensors Network System using Fuzzy Control,” *WSEAS TRANSACTIONS ON SYSTEMS AND CONTROL*, vol. 17, pp. 347–358, Aug. 2022, doi: 10.37394/23203.2022.17.39.

[29] “Triangulation.” https://en.wikipedia.org/wiki/Triangulation

[30] “Bearing (angle) - Wikipedia.” https://en.wikipedia.org/wiki/Bearing_(angle)

[31] E. Vasilaki *et al.*, “Inertia Sensor Detecting Materials using Electromagnetic Signals,” *WSEAS TRANSACTIONS ON SYSTEMS*, vol. 21, pp. 140–146, Aug. 2022, doi: 10.37394/23202.2022.21.15.

[32] E. Vasilaki and E. Antonidakis, “Medicine detection with Low Frequency Electromagnetic Signals,” *WSEAS TRANSACTIONS ON BIOLOGY AND BIOMEDICINE*, vol. 17, pp. 99–103, Sep. 2020, doi: 10.37394/23208.2020.17.12.

[33] G. Ben-Artzi, H. Hel-Or, and Y. Hel-Or, “The Gray-Code Filter Kernels.”

[34] R. W. Doran, “CDMTCS Research Report Series The Gray Code The Gray Code,” 2007.

[35] J. R. Bitner, G. Ehrlich, and E. M. Reingold, “Efficient Generation of the Binary Reflected Gray Code and Its Applications,” 1976.

[36] H. Mehta, R. M. Owens, and M. J. Irwin, “SOME ISSUES IN GRAY CODE ADDRESSING.”

[37] A. Al-Shafi and A. N. Bahar, “Novel Binary to Gray Code Converters in QCA with Power Dissipation Analysis,” *International Journal of Multimedia and Ubiquitous Engineering*, vol. 11, no. 8, pp. 379–396, Aug. 2016, doi: 10.14257/ijmue.2016.11.8.38.

[38] “Java | Oracle.” https://www.java.com/en/

[39] “IntelliJ IDEA – the Leading Java and Kotlin IDE.” https://www.jetbrains.com/idea/

[40] “Node-RED: Low-code programming for event-driven applications.” https://github.com/node-red/node-red

[41] “Calculate distance and bearing between two Latitude/Longitude points using haversine formula in JavaScript.” https://www.movable-type.co.uk/scripts/latlong.html