TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

B-10-BAS-2019/24

Development of an An Autonomous Unmanned Aerial System for Radio Frequency Source Localization

By:
Nirajan Prasad Gupta (076BAS023)
Prithak Dahal (076BAS031)
Priyank Raj Acharya (07BAS032)
Swastik Om B.K. (076BAS045)

A PROJECT REPORT TO THE DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF AEROSPACE ENGINEERING

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING
LALITPUR, NEPAL

March, 2024

COPYRIGHT

TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
CENTRAL CAMPUS PULCHOWK
DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

LETTER OF APPROVAL

The undersigned certify that they have read, and recommended to the Institute of Engineering for acceptance, a project report entitled **" Development of an An Autonomous Unmanned Aerial System for Radio Frequency Source Localization "** submitted by **Nirajan Prasad Gupta, Prithak Dahal, Priyank Raj Acharya and Swastik Om BK** in partial fulfillment of the requirements for the Bachelor's Degree in Aerospace Engineering.

Supervisor: **Arun Bikram Thapa**
Assistant Professor
Department of Mechanical and Aerospace Engineering
Institute of Engineering, Pulchowk Campus

External Examiner: **Samrat Pradhan**
FSSD
Civil Aviation Authority of Nepal

Head of Department : **Sudip Bhattarai (PhD)**
Assistant Professor
Department of Mechanical and Aerospace Engineering
Institute of Engineering, Pulchowk Campus

Date: 13th March,2024

ABSTRACT

Radio-based direction finding or radio-based localization is a new and growing field of study for an Unmanned Aerial System. While this concept had been used in manned aircrafts previously, the 'autonomous' process in UAS makes it an interesting area of research. Although there are many approaches to localization in outdoor environments, the proposed project primarily aims to utilize both the deterministic and probabilistic approach to localize the radio source in an outdoor environment with an autonomous drone, utilising a moxon antenna and an omni-directional antenna.

ACKNOWLEDGEMENT

We are grateful to the Department of Mechanical and Aerospace Engineering for giving us the opportunity and platform to conduct this project.

We would like to express our sincere gratitude towards our supervisor Asst.Prof.Arun Bikram Thapa and Asst.Prof.Dr. Sudip Bhattarai for their assistance in providing ideas and assessing our project title.

We would like to thank our seniors Keshav Poudel (075AER018), Prabhav Regmi (075AER026) and Sandesh Parajuli (075AER037) for their valuable contribution in providing the necessary materials for our project.

We would also like to express our sincerest gratitude towards Assoc.Prof. Dr. Nanda Bikram Adhikari and Sr.Instr. Kamal Prasad Nepal from Department of Electronics and Computer Engineering for providing valuable resources and ideas in this project.

We would also like to express our gratitude towards Er. Jiten Thapa and Orion Space for providing assistance for antenna testing.

TABLE OF CONTENTS

TITLE PAGE i

COPYRIGHT ii

ABSTRACT iv

ACKNOWLEDGEMENT v

TABLE OF CONTENTS viii

LIST OF FIGURES xi

LIST OF TABLES xii

LIST OF ACRONYMS AND ABBREVIATIONS xiii

1 INTRODUCTION 1

1.1 Background 1

1.2 Problem statement 1

1.3 Objectives 2

1.3.1 Main Objective: 2

1.3.2 Specific Objectives: 2

1.4 Feasibility Analysis 2

1.4.1 Economical Feasibility 2

1.4.2 Technical Feasibility 2

1.4.3 Operational Feasibility 3

1.5 Hardware and Software Requirements 3

1.5.1 Hardware Requirement 3

1.5.2 Software Requirements 3

1.6 Scope of Project 4

2 LITERATURE REVIEW 5

3 METHODOLOGY 7

3.1 Localization Modeling 8

3.1.1 Localization with deterministic approach 8

3.1.2 Localization with probabilistic approach 9

3.2 Dynamic Modelling 13

3.3 Sensor Model . . . 14
3.3.1 Sensor Model for Moxon . . . 14
3.3.2 Sensor Model for Omni-directinal antenna . . . 15
3.4 Mathematical Model . . . 16
3.4.1 Mathematical Model for Deterministic Localization Approach . . . 16
3.4.2 Mathematical Model for Probabilistic Localization Approach . . . 16
3.5 Normalization . . . 22
3.5.1 Min-Max Normalization . . . 23
3.5.2 Z-Score Standardization . . . 23
3.6 Beliefs and Filtering . . . 24
3.6.1 Discrete Bayes' Filter . . . 24
3.6.2 Recursive Discrete Bayes' Filter . . . 24
3.6.3 Particle Filter . . . 26
3.7 UAS Development . . . 26
3.7.1 Drone's CAD Geometry . . . 26
3.7.2 UAS Hardware . . . 27
3.7.3 Hardware for transmitter and receiver system . . . 33
3.7.4 Sensing modality . . . 36
3.7.5 Antenna Fabrication . . . 38
3.8 Autonomous drone . . . 39

4 Results and Discussion **41**
4.1 Antenna Testing and results . . . 41
4.1.1 Simulation . . . 41
4.1.2 Validation
. . . 41
4.1.3 Determining Path loss Index for Omni-directioal antenna . . . 43
4.2 Results from test of Deterministic Localization . . . 45
4.3 Development of Probabilistic Localization Algorithm . . . 46
4.4 Simulation of Probabilistic Localization Algorithm . . . 47
4.4.1 Simulation Setup . . . 47
4.4.2 Overview of Simulation process . . . 48
4.5 Limitations . . . 51
4.6 Problem Faced . . . 51

5 CONCLUSIONS AND RECOMENDATIONS **52**
5.1 CONCLUSIONS . . . 52
5.2 Future Enhancements . . . 52

REFERENCES **53**

6 Appendix **55**

List of Figures

3.1 Methodology Flowchart 7

3.2 Deterministic localization approach flowchart 9

3.3 drone with octant numbering 10

3.4 Probabilistic localization approach flowchart 11

3.5 Rotation of moxon antenna by yawing drone 13

3.6 Drone heading/front center line with respect to magnetic north 13

3.7 Geometric representation of relative bearing. 15

3.8 Rotation of Moxon to obtain RSSI signals at different bearings to target . . 18

3.9 Direction estimate 19

3.10 Estimation of distance from initial point T_0 20

3.11 Calculation of 1^{st} time-step as 0.65 of estimated time from T_0 21

3.12 Calculation of 1^{st} time-step as 0.65 of estimated time from T_1 21

3.13 Calculation of 2^{nd} time-step as 0.65 of estimated time from T_2 22

3.14 Localization for source 22

3.15 Localization near source 25

3.16 CAD model of the drone 26

3.17 Final assembled drone with receiver system 27

3.18 PM07 28

3.19 Pixhawk 28

3.20 Motor 29

3.21 Propellers 30

3.22 ESC 30

3.23 Battery 31

3.24 Raspberry pi 4B 32

3.25 M8N GPS Module 32

3.26 LoRa transmitter 34

3.27 LoRa transmitter circuit diagram 34

3.28 omni receiver 35

3.29 Circuit diagram for omni receiver 35

3.30 Moxon receiver 36

3.31 Circuit diagram for Moxon receiver 36

3.32 Sensing modality 37

3.33 Moxon antenna dimension specification 38

3.34 Moxon antenna dimension visualization 38

3.35 Left: CAD Model. Right: Actual fabricated Model 39

3.36 interfacing of Raspberry pi with pixhawk for autonomy 40

4.1 Left:Radiation Pattern in Horizontal Plane. Right:Radiation Pattern in Vertical plane 41

4.2 Moxon Antenna 4th iteration, tested on a Nano VNA depicting SWR 42

4.3 Testing at campus premise 42

4.4 finding out n-value 43

4.5 Variation of RSSI with distance 45

4.6 Actual and estimated locations of the target 46

4.7 RSSI distribution over the region 47

4.8 Path traced for first case 48

4.9 Path traced for long range case 49

4.10 Path traced for best case 50

List of Tables

4.1 RSSI Measurements 44

4.2 Moxon readings at different bearings 45

LIST OF ACRONYMS AND ABBREVIATIONS

RF	Radio Frequency
GPS	Global Positioning System
RSSI	Received Signal Strength Indicator
AoA	Angle of Arrival
ToA	Time of Arrival
TDoA	Time Difference of Arrival
RSS	Received Signal Strength
DoA	Direction of Arrival
UAV	Unmanned Aerial Vehicle
UAS	Unmanned Aerial System
Li-Po	Lithium Polymer
ESC	Electronic Speed Controller
Aps	Access Points
WSN	Wireless Sensor Network
FOV	Field of View
SWR	Standing Wave Ratio

CHAPTER 1: INTRODUCTION

1.1. Background

Unmanned Aerial Systems (UAS) have gained significant popularity across diverse applications enabled by the advances and miniaturization in computing, communication, and sensing. UAVs play a significant role in both civilian and military applications ranging from remote sensing, search and rescue, to environmental monitoring, to aerial communications and networking [1].

Above mentioned applications of UAV/UAS over various areas of interest involve the idea of localization of source and sensor or transmitter and receiver at its core. Localization is usually done by various methods. Some of the most common methods involve GPS localization and RF localization techniques among others.

While GPS has become the prevailing method for location estimation, it requires a direct line of sight (LOS) so it seems unfeasible to be implemented for shadowing environments [16],[2]. Similarly in the event of a power outage, GPS signals are unavailable. Mobile devices, which are commonly used for GPS-based location information, suffer from restricted battery capacity, hindering prolonged operation.

To address these challenges, radio-based localization emerges as a promising alternative, radio frequency (RF) sources exhibit prolonged functionality with minimal power consumption, making them ideal for long-term usage. Thus, our research focuses on target localization through RF source localization, assuming the target possesses a radio frequency transmitting device.By employing localization algorithms, the target's location can be determined in real-world scenarios.

1.2. Problem statement

Nepal's diverse terrain, including rugged mountainous regions, dense forests, and remote rural areas, presents unique challenges for traditional monitoring and surveillance methods.

1.3. Objectives

1.3.1. Main Objective:

The main objective is to develop a quadrotor based autonomous Unmanned Aerial System that can recognize the radio frequency and develop a probabilistic localization algorithm to locate the source.

1.3.2. Specific Objectives:

- To customize a drone and to incorporate various sensors required for target localization.
- To develop transmitter module and trans-receiver module with a moxon antenna and omni-directional antenna .
- To develop a code for the turning mechanism of a drone after receiving the RF Signals.
- To simulate the probabilistic localization algorithm

1.4. Feasibility Analysis

1.4.1. Economical Feasibility

The project entails the integration of diverse electronic components, which constitute a substantial portion of the overall budget. To date, project expenses have been primarily funded through individual contributions from team members, yet such resources may prove inadequate. .Given the adequate funding and resources, it is noteworthy that the fabrication and application expenses associated with the drone are deemed reasonable and within manageable limits, thereby affirming the project's economic viability for continued progression.

1.4.2. Technical Feasibility

The design and manufacturing prerequisites necessary for the drone's completion are wholly supported by the resources at the Department of Mechanical and Aerospace Engineering. While certain technical specifications may not have been collectively addressed previously,

individual elements have been successfully executed. Moreover, the theoretical framework required for construction is well-established and has been covered within the curriculum for a Bachelor's degree in Aerospace Engineering. Hence, based on these considerations, the project is deemed technically feasible.

1.4.3. Operational Feasibility

The localization of the RF source does not demand an expansive or pristine environment. The space necessary for this purpose is readily available within the campus area. Additionally, the calibration of the UHF source can be effectively carried out within the available space. The primary operational challenge for the drone lies in obtaining flight permission. As per the regulations,drones weighing under 2 kg are exempt from requiring flight permission in unrestricted airspace. Given that the drone being developed for RF source localization falls within this weight category, the project is deemed operationally feasible.

1.5. Hardware and Software Requirements

1.5.1. Hardware Requirement

- Drone with GPS and flight controller for stable rotation (yaw) and hovering.
- RF receiver (Moxon antenna and omni-directional antenna) capable of detecting signals from the RF source.
- A computer or data processor to collect, process, and analyze the RF data.
- Communication system (e.g., Wi-Fi or telemetry) to transmit data from the drone to the ground station.
- microcontrollers capable of executing code for the transmission and receiving of signal.

1.5.2. Software Requirements

- Solidworks
- QGroundControl
- Mission Planner

- Mapping software (e.g., Google Maps) to visualize the location of the RF source,
- Data storing and analyzing software (Cool term and Excel)
- Putty, real VNC viewer
- Visual Studio Code
- Thonny
- Arduino IDE
- Jupyter Notebook

1.6. Scope of Project

- Usage of UAVs can reduce time and cost for localization as compared to other means such as helicopters.
- Autonomous UAVs are effective for difficult and hazardous terrains for search and rescue operation.
- UAVs can be deployed to locate distress signals from missing persons, downed aircraft.etc helping rescuers narrow down search areas and improve response times
- RF source localization can provide valuable data for urban planning and management initiatives by mapping out RF signal coverage areas, identifying dead zones or interference sources, and optimizing the placement of communication infrastructure to improve connectivity and service quality in urban environments.

CHAPTER 2: LITERATURE REVIEW

Location estimation can be done by the measurements of several parameters. Target localization and node self-localization are important components of node positioning.

Target localization refers to the process of identifying the position or location of a target object within a given environment. Target localization calculates the target position from measurements made by several noisy sensors. While self-localization refers to the process by which individual nodes or entities in a network or distributed system determine their own positions or coordinates within the network.

A node self-localization can be broadly categorized into range-free and range-based methods. [3]

Range-free localization is an approach for calculating node position utilizing the distance between the transmitter and reception nodes. The techniques employed are centroid, DV-hop, and geometry conjecture [4].

Range-based localization schemes utilize inter-device angles or device-to-device distances to determine the distances between anchor nodes and sensor nodes. The availability of point-to-point distance or angle information is a prerequisite for range-based localization [5]. Subsequently, they estimate the location by employing various geometric techniques. The positions of anchor nodes are already known, either through GPS or manual pre-programming during deployment. On the other hand, sensor nodes compute their location by referencing the anchor nodes(6).

Time of arrival (ToA), time difference of arrival (TDoA), received signal strength indicator (RSSI), and angle of arrival (AoA) etc are some examples of range-based approach [6],[7]. Though ToA and TDoA have been shown to produce more precise location predictions, they are particularly susceptible to timing errors and their implementation needs expensive hardware [8]. Similarly, RSSI is one of the widely recognized range-based methods, and is easy to implement, but is prone to multi-path and signal fading and shadowing effects [9]. .

The AOA model localizes targets in a passive manner, which is highly desirable in many sensor network applications. Location derivation using bearing-only measurements[10],[11] and bearing-distance hybrid measurements [12],[13] has been an active research area. Besides position location, AOA measurements introduce potential improvements in terms of larger range coverage, reduced multi-path fading, and lower power consumption by directional transmission and interference suppression [14].

Earlier efforts to measure radio frequency signals relied on mobile robots to create maps of signal intensity over a large region[15]. This method was later applied on an unmanned aerial vehicle for radio-tagged sturgeon localization, assuming a stationary radio source transmitting with constant strength [16]. This approach, however, is inefficient since the vehicle must traverse the whole region to build a signal strength map.

When a directional antenna is pointed exactly towards the source, it has the maximum gain. Using this characteristic, the rotate-for-bearing method may be utilized to approximate the bearing from a series of strength estimations. When a directional antenna is turned, it senses maximum strength at a certain heading, which is the estimated bearing to the source[17] .

The antenna can be rotated using a dedicated actuator[12] or, in the case of multirotor UAVs, by rotating the entire vehicle [18],[19]. The received signal strength measurements are influenced by uncertain factors such as transmitter power and distance. Complete rotations are necessary to standardize the measurements and determine the estimated bearing. These procedures are slow and power expensive to a great extent.

Incorporation of another sensor, omni-directional antenna, can eliminate the need to perform actuator assisted or rotatory calibration [5].

In this work, we present a system designed to address these flaws. Our system is based on comparing the signal strength received by two antennas carried by a multirotor UAV—one mounted facing forward, the other facing back. If the forward-facing antenna receives higher strength, the RF source is deemed to be in front of the UAV and vice versa. Combining these measurements with the UAV's GPS position and heading, an estimate of the source's location is produced. In the onboard computer, a greedy, information-theoretic planner uses this estimate and leverages the UAV's maneuverability to provide velocity commands. [20]

CHAPTER 3: METHODOLOGY

The methodology of this project will be based on the following flowchart.

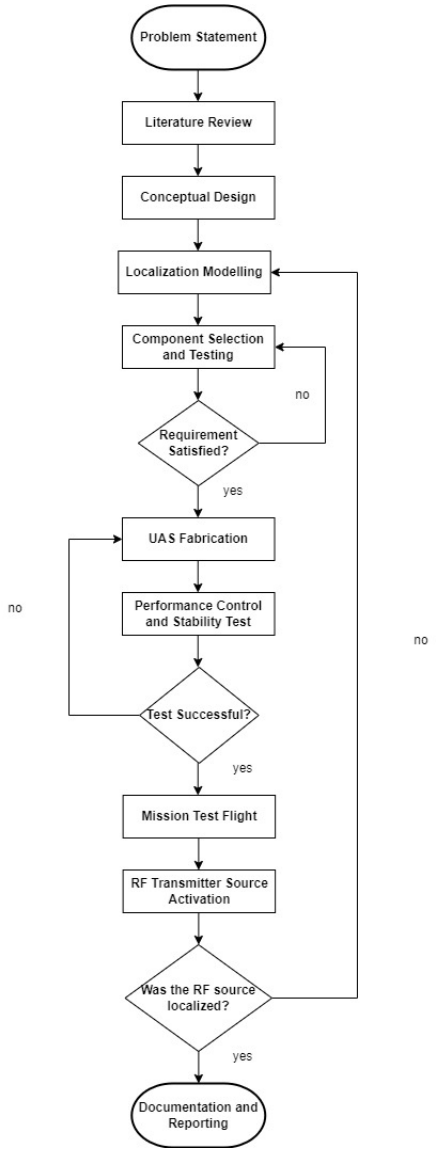

Figure 3.1: Methodology Flowchart

3.1. Localization Modeling

The aim of this project is to use a UAV to locate a stationary RF source with minimum error. The UAV equipped with a moxon antenna and an omni-directional antenna, will process the radio frequency signals to localize RF-source's existence inside a certain area or space.

Our work involves localization using two approaches:Deterministic and Probabilistic. It is to be noted that we have formulated separate mathematical models and subsequently localization algorithm for these approaches although sensor model and dynamic model are same, constrained by the fact that both share same hardware implementations.

3.1.1. Localization with deterministic approach

In this approach, possible direction towards RF source is given by bearing corresponding to maximum RSSI value. The UAS travels towards this heading for a selected time, calculated from distance approximation by using path-loss model (discussed in sensor model for omni-directional antenna).

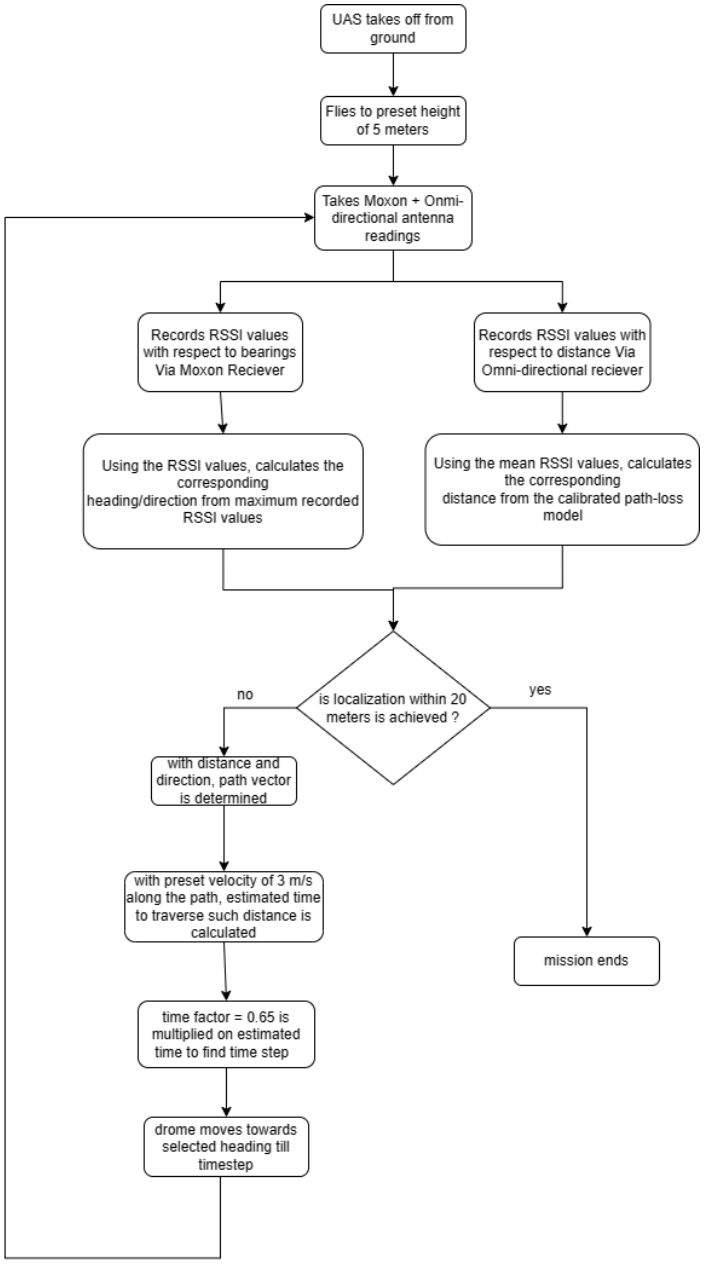

Figure 3.2: Deterministic localization approach flowchart

3.1.2. Localization with probabilistic approach

The possible directions leading to RF source is given by 8 vectors originating from drone's own position, 45 degree apart from each other. The rest of this report will refer the collection of above mentioned vectors as Octants, being angular bisections of quadrants of a 2D plane. Our goal is to achieve dependable positioning accuracy using a single UAV by concentrating belief, with respect to expected observation for these octants.

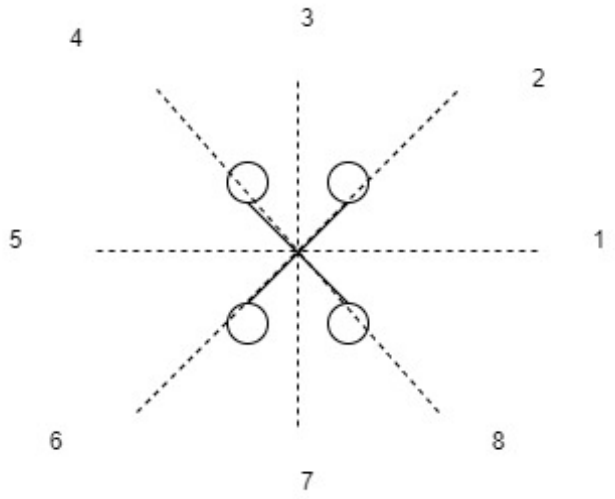

Figure 3.3: drone with octant numbering

GPS and magnetometer data will be incorporated to establish the position of the drone itself in the space.

Following method is employed to locate the target RF-source:

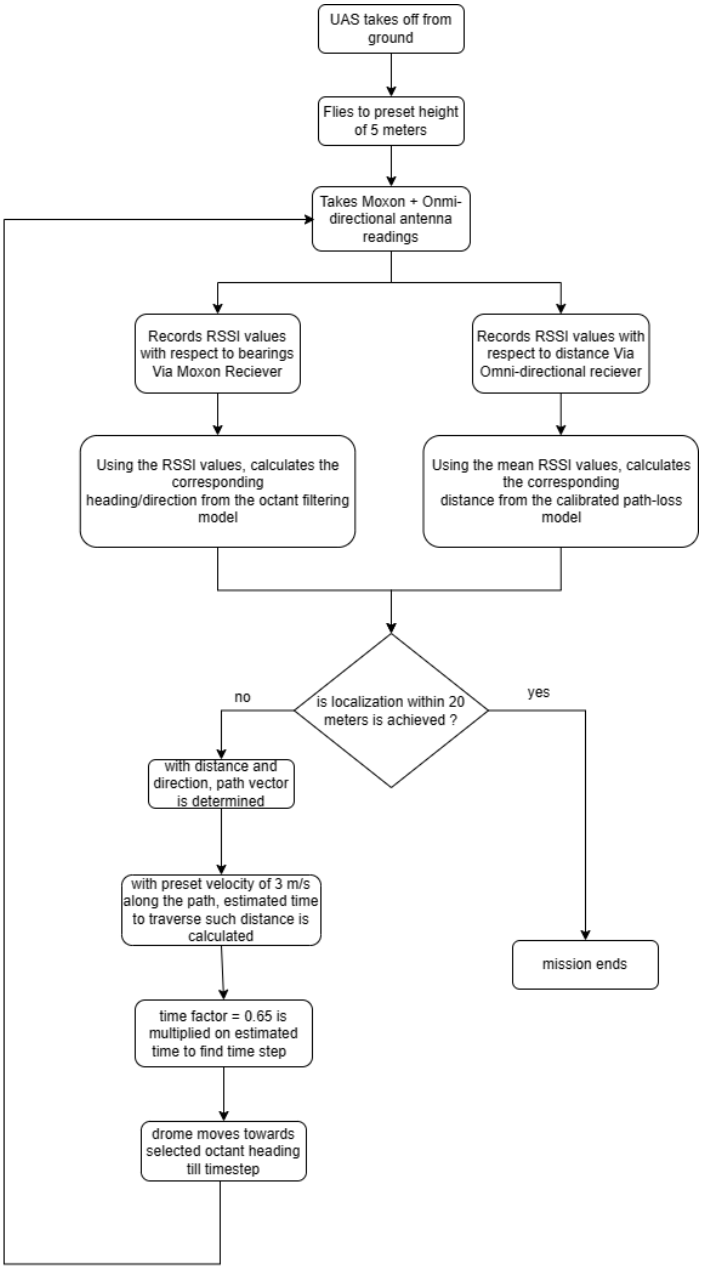

Figure 3.4: Probabilistic localization approach flowchart

The depicted flowchart illustrates the operational procedure of our Unmanned Aerial System (UAS), employing Received Signal Strength Indicator (RSSI) data to accomplish localization within a defined radius of 20 meters. Upon initiation, the UAS ascends to a predetermined altitude of 5 meters. Subsequently, it utilizes Moxon and Omni-directional antennas to capture RSSI values indicative of angular orientation and distance respectively. These measured values are then utilized to deduce heading and distance from a calibrated path-loss model. In cases where localization within the specified 20-meter radius is not attained, the UAS calculates a path vector based on distance and direction data. It then estimates the time required to traverse each segment at a preset velocity of 3 m/s, adjusts this estimation with a predetermined time factor of 0.65 to determine the time step, and proceeds towards the designated octant direction until the stipulated time step is reached. Upon successful localization within the designated range, the mission concludes. This methodological process ensures the precise localization of the UAS, a pivotal aspect for its effective operation in various engineering applications.

3.1.2.1 Regarding use of Moxon Antenna in probabilistic-belief model

The sensing with Moxon antenna involves the manipulation of the antenna's gain pattern to focus on a particular direction out of above mentioned 8 octants. While Moxon antennas themselves have a relatively wider directionality-lobe compared to some other antennas, belief generation and filtering techniques can resolve these limitations. Sensing can be achieved by mechanically altering the antenna's configuration by rotating UAV's orientation in-flight, recording new measurements at new location and heading. Recursive Belief filtering can then be done which will lead to belief concentration and ultimately to localization.

3.1.2.2 Regarding use of Omni-directional Antenna in probabilistic-belief model

Upon concentrating belief towards a single heading by processing signal acquired from Moxon antenna, the UAV moves and our next concern is providing distance of that path. This is determined by acting upon RSSI values obtained by omni-directional antenna. A distance conversion model based on the Received Signal Strength Indicator (RSSI) values is implemented. The dynamic velocity of drone is selected, which when assumed uniform, will give time value. Certain proportion of that estimated travel time is only pursued as next sensing iteration is carried out from there. Subsequent sections of this chapter will explore these concept in detail.

Figure 3.5: Rotation of moxon antenna by yawing drone

3.2. Dynamic Modelling

The mathematical formulation of dynamic modelling of drone and source is based on 2D grid of real numbers such that x_t , $\theta_t \in \mathbf{R}^2$, where:

$$x_t = [x_t^n, x_t^e, h_t]^T,$$
$$\theta_t = [\theta_t^n, \theta_t^n]^T.$$

Here, x_t^n and x_t^e represent the north and east components of the drone position. The drone heading, denoted h_t, is measured east of north and defines the direction the drone faces established by choice of front moxon. Likewise, θ_n and θ_e represent the north and east components of the radio source position. Drone state/Mathematical Model omits altitude reference eliminating the need of modelling and subsequent computation in 3D domain. This is based on the fact that gain over elevation (angle/range/level) is constant after obtaining physical and simulation validation. This will resolve vulnerability to uncertainty in RF source altitude.

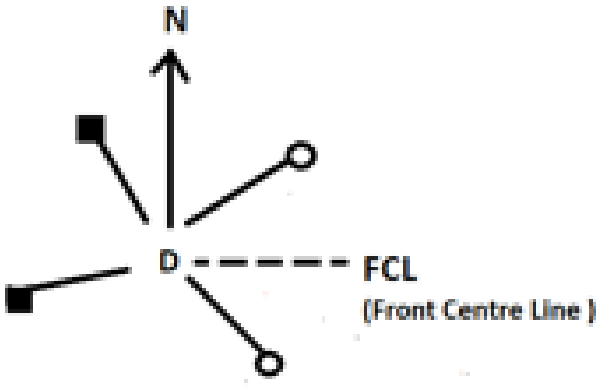

Figure 3.6: Drone heading/front center line with respect to magnetic north

For control or dynamic movement we assume Deterministic, Single integrator dynamics(linear) such that planar and rotational speed of drone are controlled directly upon actuation/initiation assuming time spent accelerating to commanded velocity is negligible. In our project, the movement velocity is selected as 3m/s.

Essentially, Planar speed of drone is f(x_t^n , x_t^n ,t) and rotational speed of drone is g(h_t,t). Both functions f and g can be condensed to obtain a dynamic state space u_t of drone .This follows to: $u_t = [\dot{x}_n^t, \dot{x}_e^t, \dot{h}^t]$, where dot($\dot{}$) above a variable indicates temporal derivative. Above formulation leads to $x_{t+\Delta t} = x_t + u_t \Delta t$. Since, x_t, u_t is completely known, new position $x_{t+\Delta t}$ after time elapsed Δt is known. This establishes determinism.

Since, $\Delta t_{\text{RFS}} >>> \Delta t_{\text{D}}$ or Δt_{SSR}, RF Source is assumed stationary: so that $\dot{\theta}_e = \dot{\theta}_n = \dot{\theta}_t = 0$. Here, t sub-scripted RFS, D and SSR represent time elapsed for change in position of Radio Frequency Source, Drone and Sensor Sampling Rate respectively.

3.3. Sensor Model

3.3.1. Sensor Model for Moxon

Our sensor model is a probabilistic model that defines the probability of making measurement z_t at time t, given drone state x_t and the radio source location Θ_t . Therefore, the probability in conjunction with Bayes' theorem is represented as $P(z_t|x_t, \Theta_t)$. As the drone moves during localization, Bayes' theorem of conditional probability updates the distribution of possible radio source location. The collection of potential observations is represented by z . It is assumed that measurements are received at intervals of seconds, aligning with the frequency at which commands are issued to the drone.

The bearing β_t is defined as $\beta_t = \arctan\left(\frac{\theta_t^e - x_t^e}{\theta_t^n - x_t^n}\right)$ where the angle is measured east of north, of a ray pointing from the drone position to the position of the radio source. The difference between this bearing value and heading of drone itself is defined as relative bearing, mathematically given as $\beta_t - h_t$. This instantly deduces to the relative bearing being 0° when the front center-line of drone points directly at the radio frequency source.

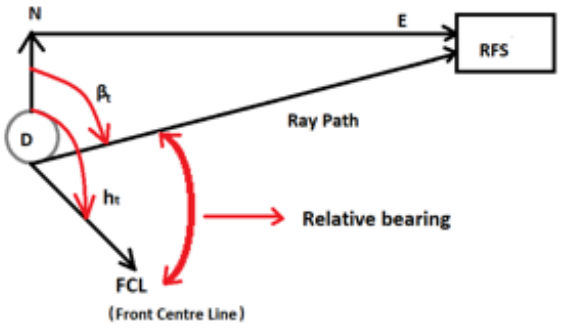

Figure 3.7: Geometric representation of relative bearing.

3.3.2. Sensor Model for Omni-directinal antenna

RSSI values are the measure of the power of received radio signals measured in dBm. In RSSI-based localization, a receiver will measure the RSSI values of the signal transmitted by the transmitter. RSSI values (measured in dBm) give the power of the received signals as the signal propagates from transmitter to receiver. As the signal power decays with distance, RSSI values can be used to infer the proximity between the receiver and transmitter by approximating the distance between them. One of the most widely used distance estimation model for radio wave propagation is the log-normal model given by equation as:

$$P_L(d) = P_L(d_0) + 10n\log_{10}\left(\frac{d}{d_0}\right) + X\sigma \tag{3.1}$$

Here, P L(d) is the path loss at a distance d, d_0 is the reference distance, PL_{d_0} is the experimentally measured average path loss at d_0, and n is the path loss index. $X(\sigma)$ is the Gaussian distributed random variable with mean, $\mu = 0$, and some finite value of the standard deviation, σ. Above equation in terms of RSSI can be written as:

$$\text{RSSI} = \text{RSSI}_0 - 10n\log 10\left(\frac{d}{d_0}\right) - X\sigma \tag{3.2}$$

The effect of $X(\sigma)$ can be removed to some extent by filtering. If we neglect $X(\sigma)$, and take the reference distance d_0 equal to 1 meters then, equation becomes

$$\text{RSSI} = \text{RSSI}_0 - 10n\log 10(d) \tag{3.3}$$

$RSSI_0$ can be experimentally determined, and the value of n can be computed by minimizing the sum of squares.(refer chapter 4 for RSSI collection and estimation of path loss model)

$$f = \sum (RSSI - RSSI_0 + 10n\log_{10}(d))^2 \tag{3.4}$$

Here, f is the function to minimize. Then, the distance between the receiver and the transmitter can be approximated by

$$d = 10^{\frac{\text{RSSI}_0 - \text{RSSI}}{10n}} \tag{3.5}$$

3.4. Mathematical Model

3.4.1. Mathematical Model for Deterministic Localization Approach

The mathematical model of deterministic localization model consists of utilizing Received Signal Strength Indicator (RSSI) data to accomplish localization within a defined radius of 20 meters. Upon initiation, the UAS ascends to a predetermined altitude of 5 meters. Subsequently, it utilizes Moxon and Omni-directional antennas to capture RSSI values indicative of angular orientation and distance, respectively.

The RSSI values captured from moxon antenna are stored in an array of size 11, indicating 12 divisions of 360° field around UAV and discarding 12^{th} element (360° being redundant with 0°). This array is processed to spit out, bearing with maximum RSSI value and the heading for UAV movement is set.

The measured values from omni-directional sensor are then utilized to deduce distance from a calibrated path-loss model (refer 4.1.5). Model then estimates the time required to traverse each segment at a preset velocity of 3 m/s, adjusts this estimation with a predetermined time factor of 0.65 to determine the time step, and proceeds towards the designated heading until the stipulated time step is reached. Upon successful localization within the designated error range, the mission concludes.

3.4.2. Mathematical Model for Probabilistic Localization Approach

Our system has 2 antennas, each of them can receive RF signal from a single transmitter simultaneously and analyze those signals independently. The need to process RF signal and condense meaningful data for both moxon and omni-directional antenna is felt. We adopt following scheme.

3.4.2.1 Mathematical model for moxon antenna

First of all, we gather RSSI data at different bearings by measuring signal strength while rotating the moxon receiver by yawing the drone in flight (refer appendix for relevant yawing code). We note RSSI value corresponding to each bearing angle. Then, we normalize the RSSI values since they are in different scales. Normalization (discussed in section 3.5) helps in better visual comparison. Using a python program(refer appendix I) a polar plot containing gain pattern of moxon receiver based on normalized values versus bearings can be obtained, upon visually inspecting such gain pattern, peak distribution for top 3 octants can be estimated.

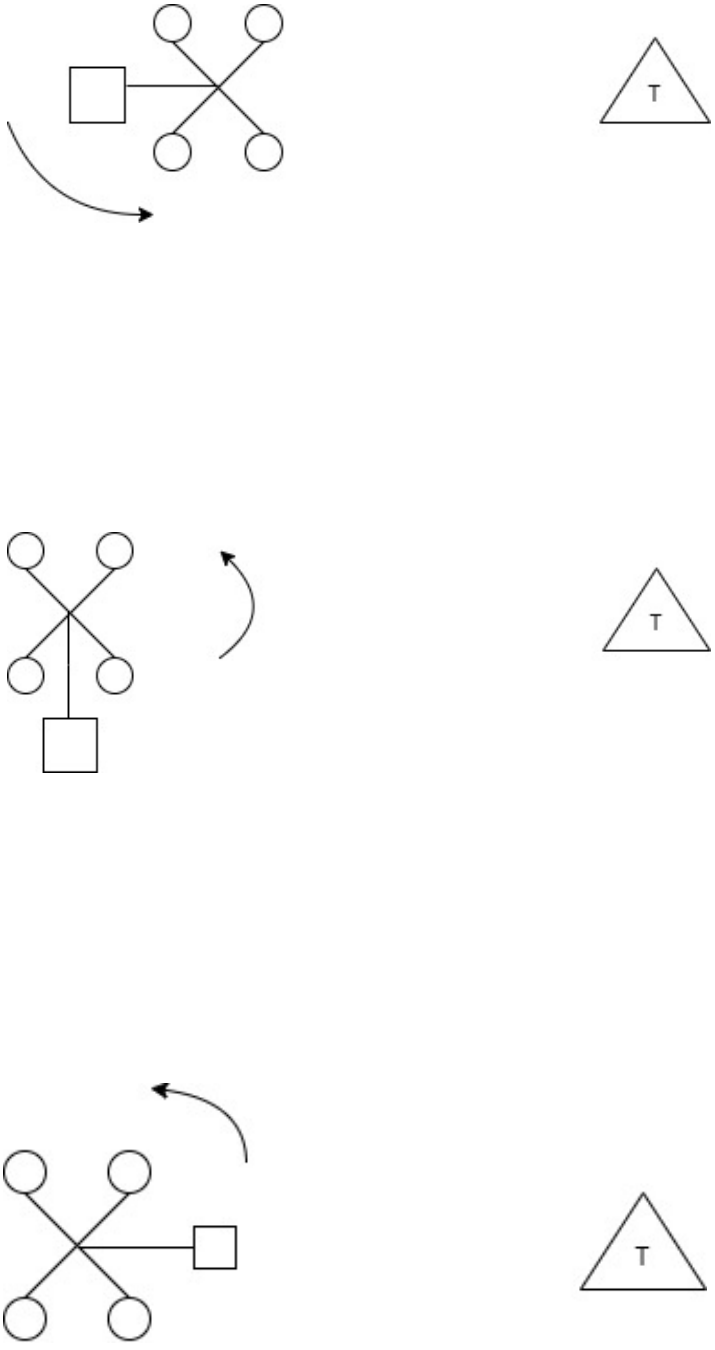

Figure 3.8: Rotation of Moxon to obtain RSSI signals at different bearings to target

This moxon data is recorded in flight and is referenced by raspberry pi for post-processing to get heading estimate of target. The raw data is imported by code (refer Appendix), it is then normalized. Out of large range of data the values corresponding to octants(1 to 8) is considered for heading determination purposes. The mathematical model supposes the target may lie along any of these 8 directions. So, probability of 0.125 is supposed along each octant for being a feasible 'direction to target'. The updating, filtering and synthesis of this belief is done via pathway filter which will be discussed in section 3.6.

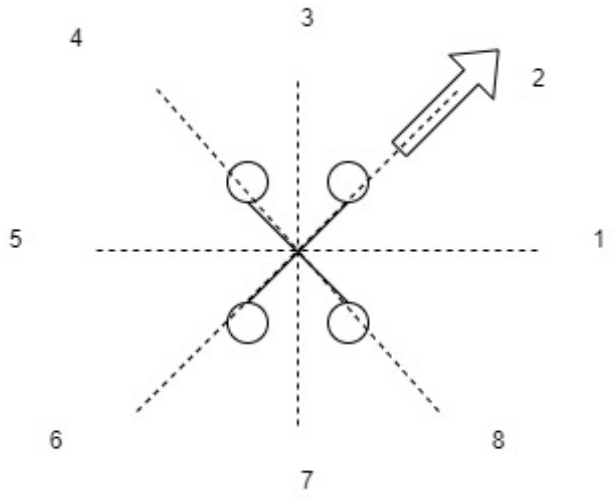

Figure 3.9: Direction estimate

3.4.2.2 Mathematical model for Omni-directional antenna

After filtering, we get direction estimate, now the RSSI data from omni-directional antenna(collected simultaneously with moxon when drone is in rotation) is used to calculate distance to the target. Since, omni-directional reading is susceptible to distance variation, we have taken extra precautions to minimize errors due to this cause, by placing omni directional receiver very close to drone's center of rotation(refer fig ....).

A matlab code was developed (see appendix) to visualize variation of RSSI with distance, value of path loss index (n)=2.17 was obtained. By using following equation

$$d = 10^{\frac{\mathrm{RSSI}_0 - \mathrm{RSSI}}{10n}} \tag{3.6}$$

with n= 2.17, $RSSI_0$= -53, distance to the RF transmitter from UAS could be estimated.

3.4.2.3 Iterative time-steps approach

The need for this iterative approach lies in fact that due to noise in RSSI measurement,

the estimated direction and distance may be off target. Thus iteratively progressing towards target, collecting measurements along the way and subsequently updating belief is desirable in attempts to minimize localization error. The end constraint that stops this iterative process is target being within 20 meters of drone vicinity, calculated from omni-directioinal antenna RSSI value.

With 3m/s constant velocity being assumed as drone speed, time required to travel distance estimated by path loss model was calculated. We iteratively reach towards target by heading along estimated direction till a certain time, (referenced as 'time-step' for remainder of this report) such that UAS progressively reaches towards target. By taking a factor of 65 percent of calculated time, time step to next point in our iterative localization is obtained. The drone moves till time-step point and halts to iterate localization process again, where it rotates to gain data for both moxon and omni-directional antenna. The factor of 65 percent is taken at random, any other factors can be chosen;lower factor corresponds to slower convergence and higher factor is susceptible to overshoots.

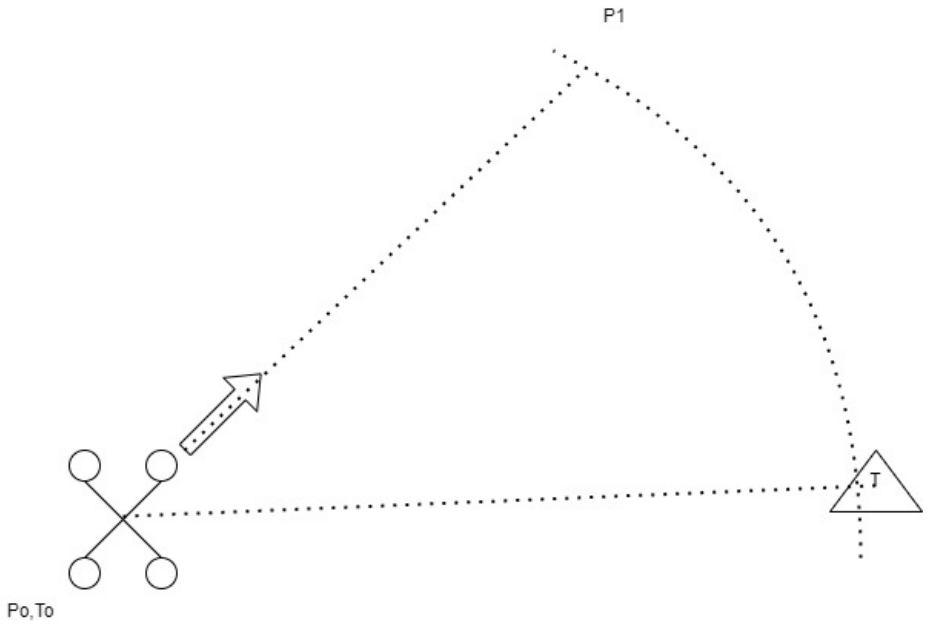

Figure 3.10: Estimation of distance from initial point T_0

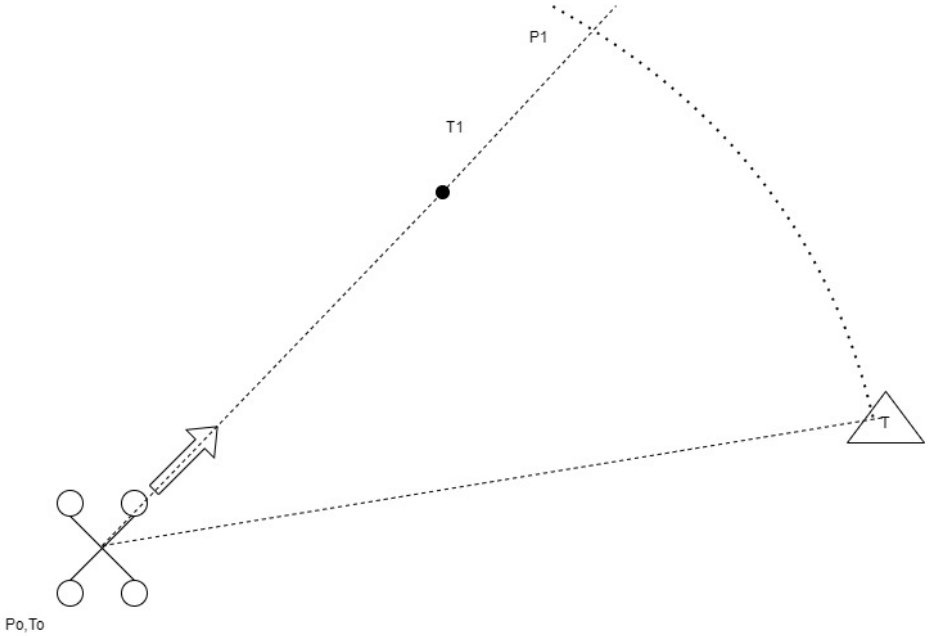

Figure 3.11: Calculation of 1^{st} time-step as 0.65 of estimated time from T_0

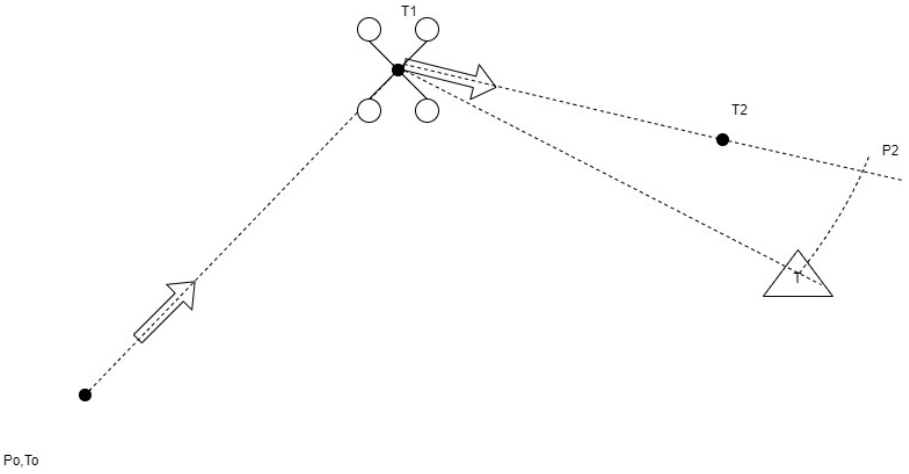

Figure 3.12: Calculation of 1^{st} time-step as 0.65 of estimated time from T_1

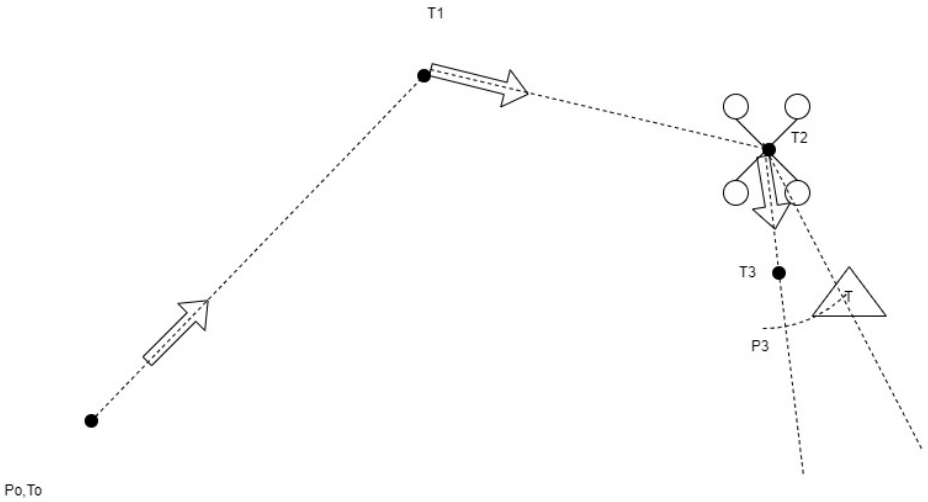

Figure 3.13: Calculation of 2^{nd}time-step as 0.65 of estimated time from T_2

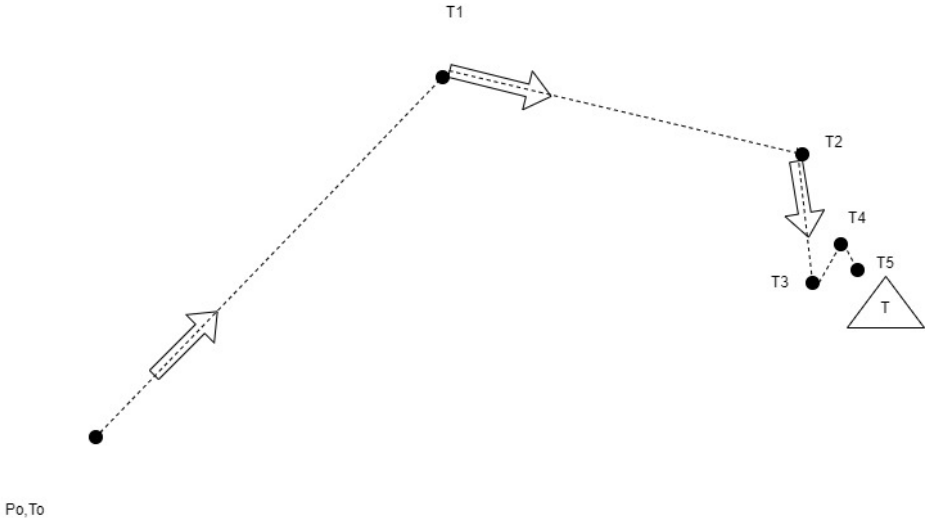

Figure 3.14: Localization for source

Diagrams above show conceptual representation of aforementioned iterative process. Simulated and actual localization iterations with relevant data and diagrams are included in subsequent sections of this report.

3.5. Normalization

Normalization is a crucial aspect of data analysis in localization .It eliminates redundancy by structuring data and removing effects of duplicate information, thereby ensuring efficient storage and avoiding data inconsistency. The risk of anomalies such as insertion, deletion, and update anomalies is also mitigated by normalization. Additionally, normalized databases

lead to enhanced scalability, ultimately culminating towards reliable insights for informed decision-making and belief updating through data analysis. Some of the common normalization techniques are:

3.5.1. Min-Max Normalization

This method scales the values linearly between a minimum and maximum value to produce normalized values between 0 and 1. . The formula for min-max normalization is:

$$X_{\text{normalized}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

where:

- X is the sprcific RSSI value.
- X_{min} is the minimum value of RSSI in data-set.
- X_{max} is the maximum value of RSSI in data-set.

This normalization technique is suitable in our localization as it is less computationally demanding when compared to other techniques like Z-score standardization, where quantities like mean, median, standard deviation etc are required . The calculation of these parameters comes with computational burden especially for systems with large data sets and weak processing chip-sets.

3.5.2. Z-Score Standardization

This method scales the values to have a mean of 0 and a standard deviation of 1. The formula for z-score normalization is:

$$X_{\text{normalized}} = \frac{X - \mu}{\sigma}$$

where:

- X is the original RSSI value.
- μ is the mean of the RSSI values in data-set.
- σ is the standard deviation of the RSSI values in data-set.

3.6. Beliefs and Filtering

Firstly, the drone maintains a belief, which helps in accounting the potential location of the radio source. Here, the belief at time t is represented by b_t. In aerospace applications, beliefs are modelled by using Gaussian distributions. For modelling by Gaussian distribution, Kalman filters are widely used. However, there are limitations to using Kalman filters. Kalman filters can be used if the models are linear or easily linearizable and distributions are uni-modal.

Hence, since our sensing techniques are expected to result in strongly non-Gaussian beliefs, this project employs non-parametric representation for beliefs: a discrete Bayes' filter for stationary radio source. The model, assumes search area mainly as a square where the initial belief assume a uniform distribution which implies that radio source has equal possibility of being located anywhere within the search area.

3.6.1. Discrete Bayes' Filter

A discrete Bayes' filter splits the search area into grids, where the density of grid gives the probability that radio source is in that cell. This method is highly preferred in drone localization as it can handle the non-Gaussian beliefs as well as non-linear dynamic and measurement models. Here θ_t represents radio source location. The belief b_t is calculated based on previous belief $b_{t-\Delta t}$, the drone state x_t and observed data z_t by applying Bayes' rule. For a stationary radio source,

$$b_t(\theta_i) \propto b_{t-\Delta t}(\theta_i) P(z_t | x_t, \theta_i)$$

where, θ_i is cell and $b_t(\theta_i)$ is probability that radio source is in the cell θ_i.

It is important to normalize the belief to ensure that total probability across all cells sums to one. For example, if grid cell has probability of 0.5, there is 50% chance, the radio source is located in that cell.

3.6.2. Recursive Discrete Bayes' Filter

Recursive Discrete Bayes' Filter is framework for recursive state estimation that utilizes Bayes Theorem, Markov assumption, Probability theory and Bayesian Networks to do so. It estimates a probability density functions over time using observations.

First of all, we define 8 octants, which we will number 1 to 8, where 2 is counter-clockwise to 1 and so on. Initially prior to measurement and any belief updates, the source is equally likely to be along any octant. There are 8 octants, so the probability that the source is along any of the octant is 1/8, as the probability density always sums to 1.

Representation of our belief at any time

A belief array is created with 8 elements. The array is initialized to 1/8 as belief distribution is assumed to be uniform. Next, we create a pathway array with all elements set to 1 representing equal weighting for each pathway. Then, we create evidence array of size 8 representing different possible pieces of evidence to be determined as per sensor readings acquired from moxon antenna. We also find the top 3 peak values from the 8 octants based on prominence and distance criteria. The position with these top 3 values are represented as 1 and all other 5 positions are represented by zero. Therefore, it updates the evidence array with 5 zeroes and 3 ones representing the presence or absence of peak of evidence.

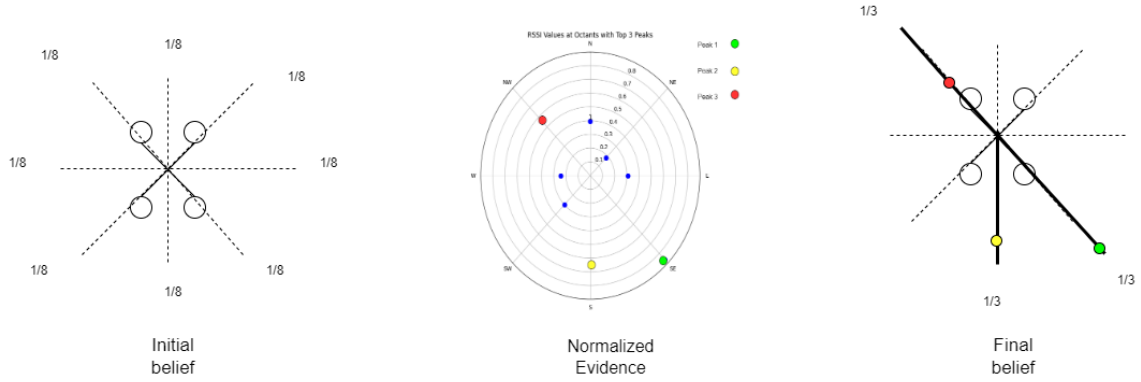

Figure 3.15: Localization near source

The belief array is calculated by element wise multiplication of evidence array with 1/3, representing likelihood of observing each peak given the pathway. There are 3 octants, so the probability that the source is along any of the octant is 1/3, as the probability density always sums to 1. Then, it adds small noise value σ to simulate measurement noise. Then, the array is normalized and belief is set based on prominence such that 1 out of 3 octant is to be selected. The system decides upon this selection by decision criteria regarding whether or not the the top 2 octant beliefs are adjacent. The adjacent beliefs support each other in deciding that target is along vicinity of these octants. The adjacent criteria ensures error in heading determination is within acceptable limit and the localization eventually converges after certain iteration. It is to be noted that iteration steps and error in each iteration is also dependent upon time factor whose selection is influenced by action execution hardware of localization UAS. Upon execution of action commands, the system is ready to generate

new belief distribution from start and synthesizes such beliefs based on evidence acquired in updated state space and subsequently new action can be taken. This process is iterative in nature and forms basis of belief handling in our project.

3.6.3. Particle Filter

Particle filter are used to overcome the limitations of discrete filter in tracking moving targets and slow computation, as computation increases exponentially with number of grid. Also, discrete filter requires target motion within the organized grid cell. Hence, while dealing with non-stationary radio source, a particle filter must be employed for effective motion modelling.

However, our project will be focused on the development and implementation of Discrete Bayes' filter only as it is sufficient for stationary radio source tracking.

3.7. UAS Development

3.7.1. Drone's CAD Geometry

A reference solidworks CAD model of the proposed drone was taken to understand the overall design and architecture.

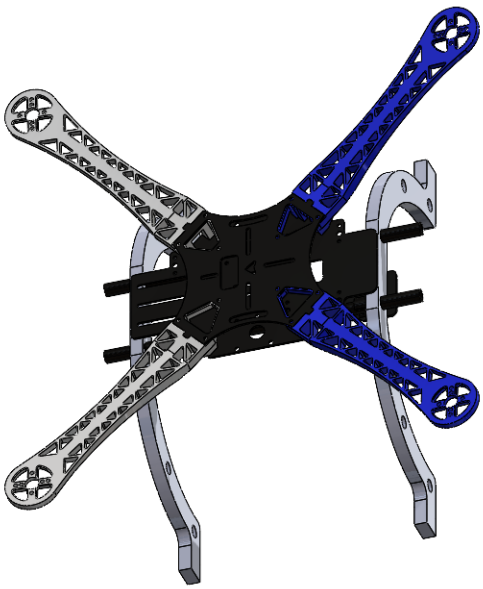

Figure 3.16: CAD model of the drone

3.7.2. UAS Hardware

The proposed drone configuration is an X-type design featuring four 1000kv DC brushless motors for enhanced performance. To ensure speed control, the configuration includes four electronic speed controllers. In order to achieve a lighter and stronger frame, a new drone frame was acquired. The flight controller of choice is the Pixhawk 4, which effectively manages and regulates the drone's operations. For flight computations and control, the Raspberry Pi 4B serves as the flight/companion computer. A 4-cell Li-Po battery powers all the components seamlessly. Propellers were selected based on the specific requirements of the mission.

Figure 3.17: Final assembled drone with receiver system

To enable precise localization and mission execution, the drone is equipped with various antennas and sensors.Once the drone is assembled, testing is done to ensure the drone is working as intended.

Various hardware were used for the drone and localization. Each of them is briefly described below:

1. **PM07**
   The PM07 is a power distribution board designed for use in drones and other remote-controlled vehicles. It efficiently distributes power from the battery to various components such as motors, ESCs (Electronic Speed Controllers), flight controllers, and other onboard electronics. With its compact design and multiple output ports, the

PM07 helps organize wiring and ensures reliable power delivery, contributing to the overall performance and stability of the drone.

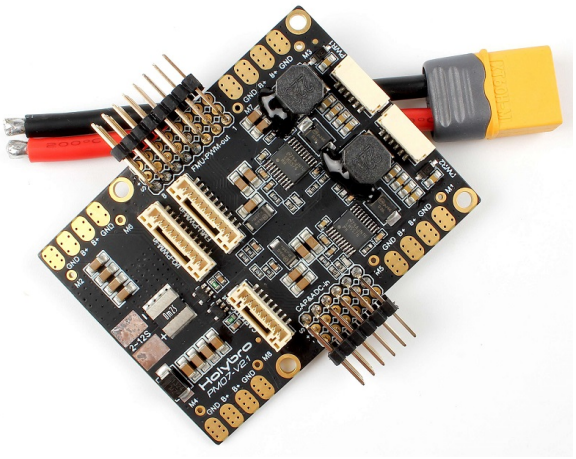

Figure 3.18: PM07

2. Pixhawk

The main flying controller that controls the quadrotor during the localization mission is Pixhawk 4. Pixhawk is used in the quadrotor configuration and is linked to the PM07, RC receiver, M8N GPS module, and Raspberry Pi 4B.

Figure 3.19: Pixhawk

3. **Motor**

The motor is the essential part of a drone since it turns the propellers to provide propulsion. To balance the force and torque generated by the propeller, the number of anticlockwise and clockwise rotating motors must coincide. The motor's size and thrust output are determined by its KV rating. A motor's KV rating is its RPM constant, which tells you how many revolutions it will make with 1 volt applied and no load attached. When a large propeller and a high KV motor are combined, the propeller will spin at a high revolution rate (RPM), which demands more torque and can lead to heating and premature failure. The drone uses four brushless 1000kv motors.

The motor is Turnigy Aerodrive SK3-3542-1000kv. It has double shielded bearings, sintered neodymum magnets, and string wound stator.

Figure 3.20: Motor

4. **Propellers**

The essential parts of the drone that produce the necessary lift force are the propellers. A propeller's size and pitch determine its performance. The moving distance covered by a propeller in a single revolution is known as its pitch. Higher pitch propellers account for more lift force, whereas lower pitch values account for more torque. Larger propellers with lower pitches are better suited for towing heavier loads, whereas smaller propellers with higher pitches are preferable for swift and agile drones. In our case, each motor has a 10*4.5 (inch) propeller attached to it. Two of the propellers that are diagonally opposed to one another rotate in a clockwise direction, and the other two rotate in an anticlockwise direction.

Figure 3.21: Propellers

5. ESC

Electronic Speed Controllers (ESCs) are essential components of drones, responsible for regulating the speed of electric motors. They respond to signals from the flight controller by adjusting the voltage supplied to the motor, thereby altering its RPM and enabling the drone to maneuver effectively. ESCs designed for drones typically specify the maximum current capacity of the motor they can handle and operate at a specific refresh rate, indicating how frequently they can adjust the motor speed per second. In the case of quadcopters, high refresh rate ESCs are commonly employed. For instance, an ESC with a 10A rating can sustain a continuous current flow of up to 10A, with the rating typically chosen higher to prevent overheating or damage. Additionally, ESCs have a burst rating, denoting the maximum short-term current they can endure without harm. In our case, four 50A ESCs are utilized for controlling the speed of the motors.

Figure 3.22: ESC

6. **Battery** The drone's battery is a vital component that powers all of its electronic components. Drones usually use Li-Po batteries that have a nominal voltage of 3.7 volts. To increase the overall voltage, the cells in these batteries are arranged in series. The letter 'S' and a numeric represent the number of cells in the battery design; for instance, 4S indicates a 4-cell setup. Battery capacity and C rating are the two main criteria taken into account when choosing a battery. Battery capacity, expressed in milliampere-hours (mAh), expresses how much current a battery can produce for a certain length of time. By raising the battery's C rating, you can increase its maximum safe current delivery capacity and use the same cell to power larger payloads.

Figure 3.23: Battery

7. **Raspberry pi**

   The Rasberry Pi 4B serves as the companion/flight computer in our project, processing RSSI data received, doing necessary computations, and generating commands to steer the drone autonomously toward the objective. The pi is equipped with a number of sensors to gauge the RSSI coming from the transmitter. The telemetry port on the Pixhawk and the power output port (5V, 3A) on the PM07 are utilized to power the Pi. The pi's serial ports are utilized to input the received signal.

Figure 3.24: Raspberry pi 4B

8. M8N GPS module

Drone applications frequently use the high-performance GPS receiver known as the M8N GPS module. It gives the drone precise GPS positioning data, which is necessary for precise flight control and navigation. A built-in compass on the M8N GPS module helps with localization algorithms. The M8N GPS module determines the drone's position by utilizing the signals it receives from several GPS satellites. The drone's flight controller receives the GPS data, which it uses for control and navigation. Because it permits a dependable and secure drone operation, the M8N GPS module is an essential part of drone applications.

Figure 3.25: M8N GPS Module

9. **RC receiver**

   The RC receiver in a drone acts as the communication interface between the pilot's transmitter and the onboard flight controller. It receives radio signals from the transmitter and relays them to the flight controller, allowing the pilot to control the drone remotely. Typically compact and lightweight, RC receivers come in various channels to accommodate different control functions such as throttle, pitch, yaw, and roll. They play a crucial role in ensuring precise and responsive control of the drone during flight. Since our drone is an autonomous one, the only purpose an RC receiver served is to trigger fail-safe. If the drone did not perform as expected, the RC would be used to overwrite the mission and make the drone either land immediately or return to the launch point.

3.7.3. Hardware for transmitter and receiver system

1. **LoRa-ESP8266 Transmitter**

   Using an ESP8266 microcontroller and an SX1278 LoRa module, a prototype transmitting device was made. when the battery is linked, the device begins sending out RSSI packets on the 433 MHz frequency band, which is a licence-free RF band for Asia.

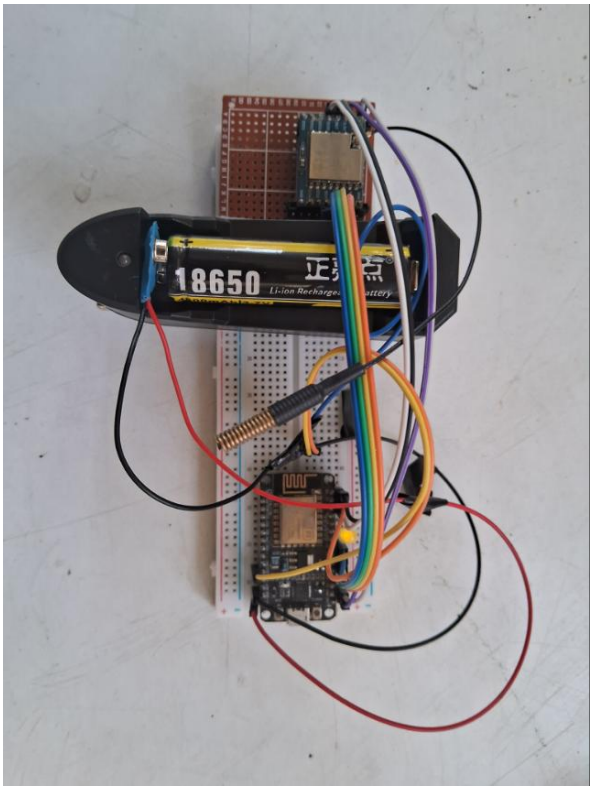

Figure 3.26: LoRa transmitter

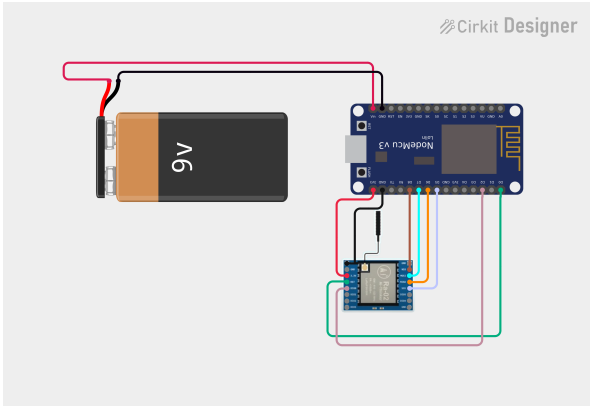

Figure 3.27: LoRa transmitter circuit diagram

Figure 3.26's circuit diagram illustrates how an ESP8266 module, a battery, and an SX1278 LoRa module are connected.

The ESP8266 module is battery-powered, and the LoRa module is connected to it via serial communication. Low-power, long-range wireless communication is enabled by the LoRa module. The ESP8266 module is a microcontroller that has integrated Wi-Fi functionality. The switch used to turn on and off the gadget is connected to the battery by the ESP8266. The ESP8266 module receives power from the battery, enabling autonomous operation. As a result, the system is portable and appropriate for usage in isolated areas.

2. **LoRa-ESP8266 receiver** Two receiving devices are built with Nodemcu esp8266 as microcontrollers. The receivers are programmed to continuously receive the transmitted signals along with their RSSI. Both receivers are discussed below along with the purpose they serve:

    - **omnidirectional receiver**

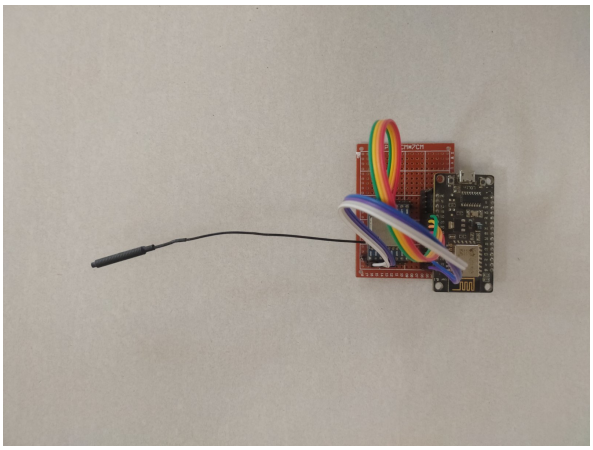

Figure 3.28: omni receiver

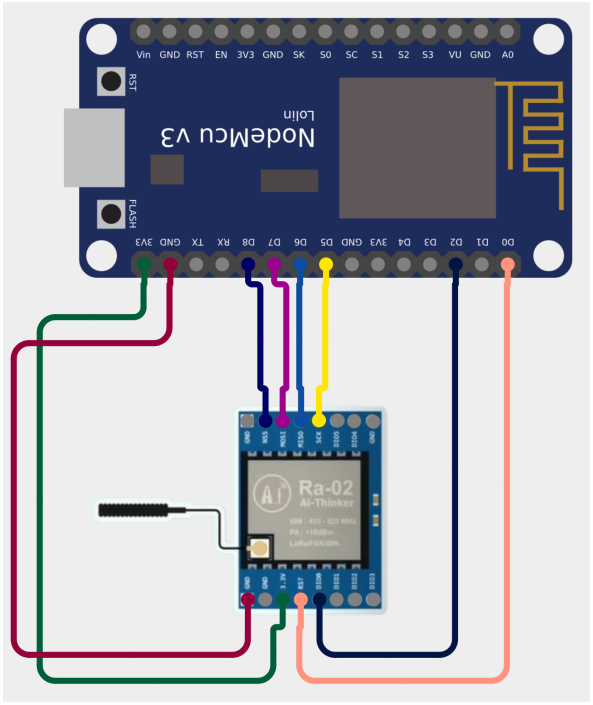

Figure 3.29: Circuit diagram for omni receiver

- **moxon receiver**

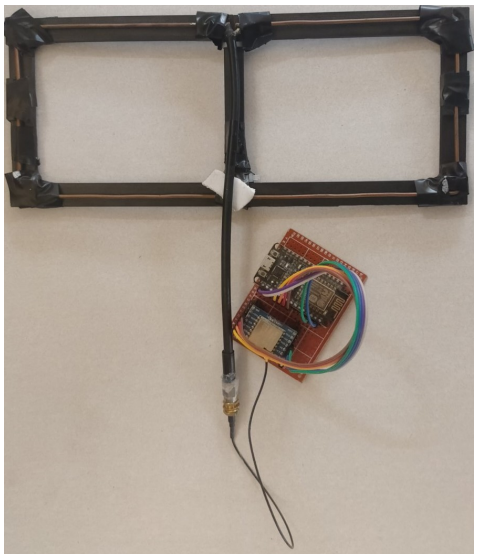

Figure 3.30: Moxon receiver

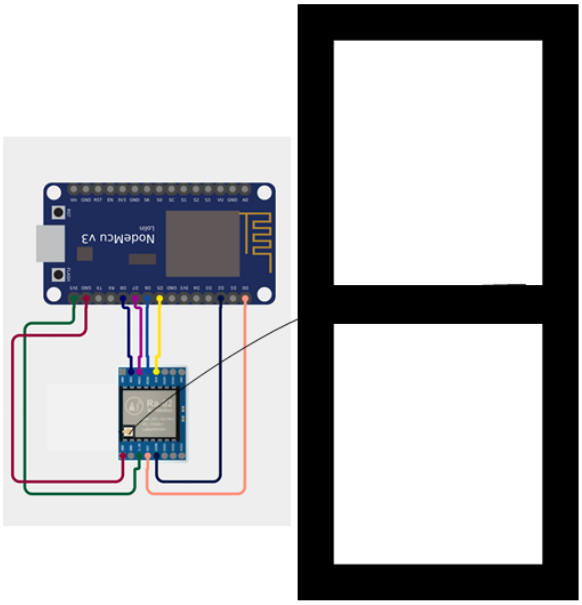

Figure 3.31: Circuit diagram for Moxon receiver

3.7.4. Sensing modality

Our localization model relies on the drone's rotation to measure RSSI values.

There are three components that make up the 433 MHz radio signal detecting mechanism:

1. The first module is an SX1278 LoRa transceiver that can measure the RF source's RSSI and is coupled to an antenna. It uses SPI to connect with the microcontroller.

2. The Nodemcu esp8266 microcontroller module is the second module. It retrieves the RSSI values from the LORa module.

3. The Raspberry Pi is the third module. It imports the RSSI readings from the modemcu, analyzes them, does the localization computation, and then instructs the Pixhawk to perform the required maneuvering.
The suggested mode of sensing is depicted in figure...

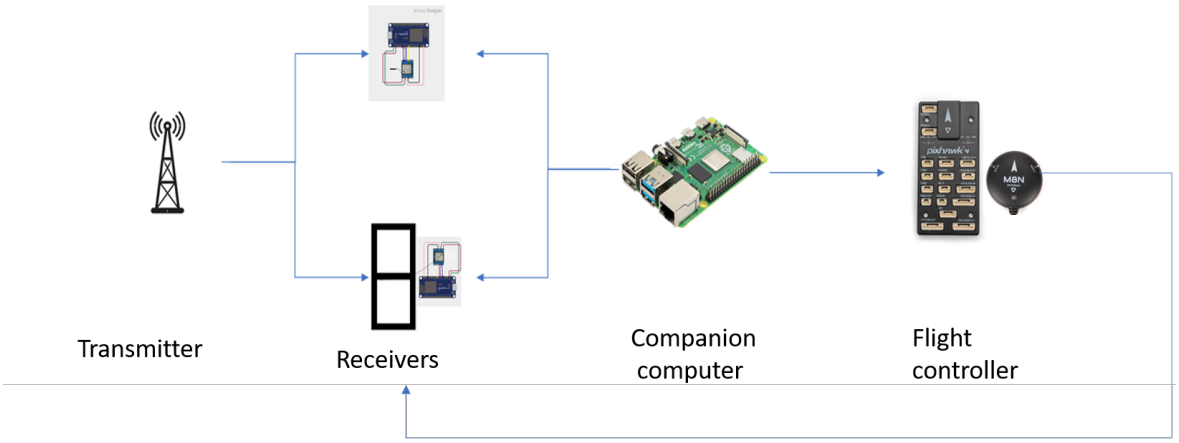

Figure 3.32: Sensing modality

3.7.5. Antenna Fabrication

The antennas used in our project are moxon antenna and omni-directional antenna.While the omni-directional antenna is bought off the shelf, the moxon antenna is fabricated by ourselves.The Moxon antenna's dimensions depend on the operating frequency (center frequency) for which it's designed which in our case is 433MHz (ISM band). The dimensions of our antenna was obtained from online tool (online moxon calculator)[].

Dimension	Wavelengths	Millimeters
Frequency 433 MHz	1.0	692.361432
Diameter 2mm	0.002889	2
A	0.340563	235.792929
B	0.05033	34.846553
C	0.009786	6.77542
D	0.071711	49.650233
E	0.131827	91.272207

Figure 3.33: Moxon antenna dimension specification

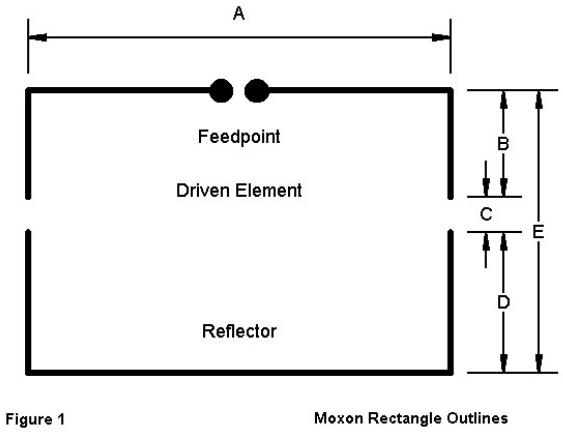

Figure 3.34: Moxon antenna dimension visualization

Copper wire of 2mm diameter is used as construction material. The outer frame of the antenna is 3d-printed with ABS filament. The antenna fabrication involved an iterative process, progressively refining each subsequent model to surpass its predecessor. The series of iterated models follows a continuous improvement trajectory, each version aims at enhancing the antenna's performance and efficiency. Fourth iteration model is selected as our final moxon model.

Figure 3.35: Left: CAD Model. Right: Actual fabricated Model

3.8. Autonomous drone

In our project, we implement localization using an autonomous drone system. This autonomy is realized through the utilization of a Raspberry Pi 4 as the flight computer. The Raspberry Pi receives sensor data, processes it, and subsequently generates commands necessary for the Pixhawk flight controller to adjust the drone's position,orientation and movement accordingly.

The libraries employed in our system are DroneKit and pymavlink.

1. DroneKit: This library provides a high-level API for interfacing with the Pixhawk flight controller. It offers functionalities to interact with the drone's flight parameters, set waypoints, and execute commands, facilitating seamless integration of autonomous features.

2. pymavlink: This library serves as a communication protocol for exchanging data between the Raspberry Pi and the Pixhawk. It enables efficient transmission of commands and telemetry data, ensuring robust communication between the flight computer and the flight controller.

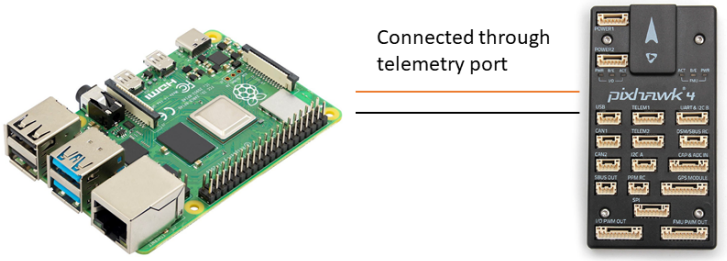

Figure 3.36: interfacing of Raspberry pi with pixhawk for autonomy

CHAPTER 4: Results and Discussion

4.1. Antenna Testing and results

4.1.1. Simulation

The antenna was modelled with the required dimensions and the radiation patterns (ideal case) in horizontal and vertical plane were obtained from 4nec2 software, shown in figure below.

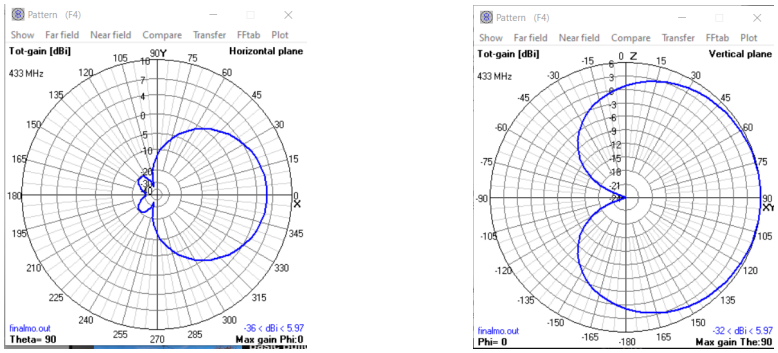

Figure 4.1: Left:Radiation Pattern in Horizontal Plane. Right:Radiation Pattern in Vertical plane

4.1.2. Validation

To verify moxon receiver's radiation and communication capabilities, our team visited Orion Space and conducted relevant testing by using Nano VNA testing module. After calibrating Nano VNA by open, short and load standards, we tested a standard 465 MHz omnidirectional antenna to establish correct tester configuration. The quantity of our interest was SWR (Standing Wave Ratio, a dimensionless quantity) which is generally acceptable if within 1-2 range. Our 4^{th} iteration moxon provided 1.83 SWR value, which was assumed to suffice for our purpose.

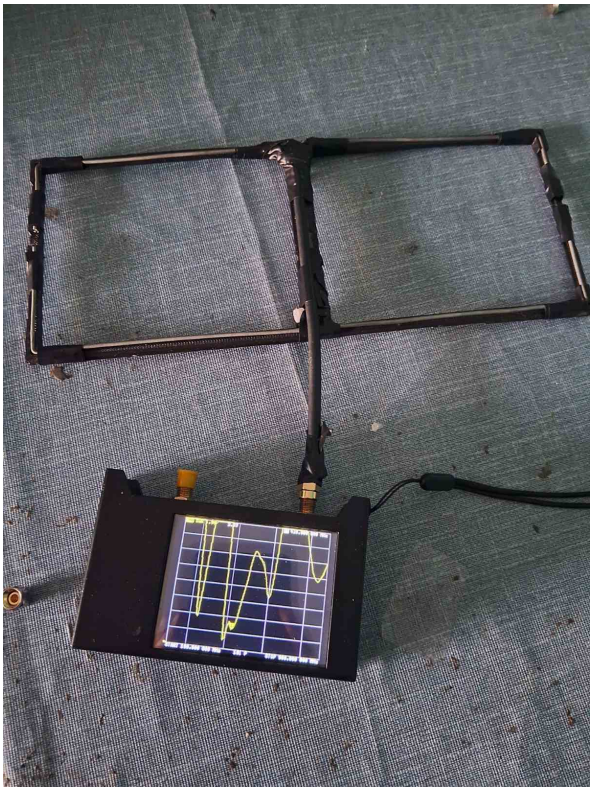

Figure 4.2: Moxon Antenna 4th iteration, tested on a Nano VNA depicting SWR

Testing of moxon antenna around campus area is shown below:

Figure 4.3: Testing at campus premise

4.1.3. Determining Path loss Index for Omni-directioal antenna

Prior to commencing the mission, it is essential to conduct a calibration test to determine the path loss exponent (n) .This involves recording RSSI values at different points and averaging them based on their respective distances from the transmitter. We conducted calibration test at pulchowk cricket ground and collected averaged out RSSI data at different distance from omni-directional reciever. The RF transmitter was standard for all distance points with same level of power throughout the experiment.

Figure 4.4: finding out n-value

Above fig illustrates distances where data was taken by white dots and red dot represents reciever location which was constant throughout the experiment. Table below depicts RSSI data at corresponding distances.

Table 4.1: RSSI Measurements

Distance (m)	**RSSI (dBm)**
1	-53
2	-54
3	-55
5	-60
10	-71
15	-79
20	-96
30	-88
40	-80
50	-90
60	-91
70	-95
80	-97
90	-99
100	-96
120	-100
130	-100
140	-95

By fitting a log-distance path loss model and minimizing the sum of squares(refer appendix for relevant code), the values of ”n”is derived as 2.17 . We note that during our testing ,the value of RSSI at 1 meter i.e. $RSSI_0$ was -54 dBm.

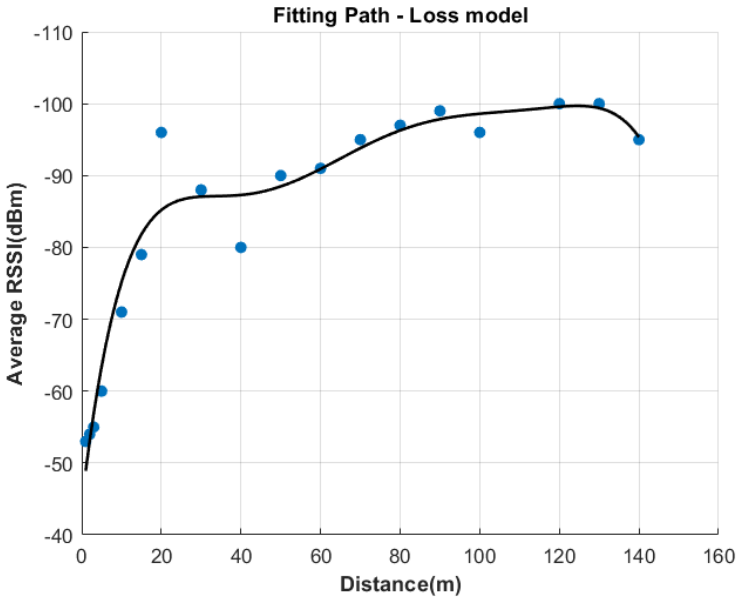

Figure 4.5: Variation of RSSI with distance

4.2. Results from test of Deterministic Localization

Table 4.2: Moxon readings at different bearings

Bearing(Degree)	Moxon(dBm)
1	-102
35	-101
60	-100
97	-101
125	-102
158	-102
183	-102
209	-102
247	-103
271	-102
303	-102

Max RSSI= -100dBm

Corresponding angle=60 degrees from north

Actual angle =76 degrees from north

Omni RSSI = -89dBm

Distance = 45.32m
Actual distance = 38m
Target location(27.683008,85.322378)
Seeker location(27.682798,85.3220805)

Figure 4.6: Actual and estimated locations of the target

Hence, from the above moxon reading table, we found our maximum RSSI to be -100dBm and corresponding angle to be 60 degrees. But, the actual angle is 76 degrees.The angular error was found to be 16 degrees.

Also, from Omni-directional antenna we found out Omni RSSI to be -89dBm and corresponding distance to be 45.32m whereas our actual distance was 38m. The distance error was found to be 7.32m.

4.3. Development of Probabilistic Localization Algorithm

A probabilistic localization algorithm based on mathematical model discussed in (section 3.4.2) was developed. The algorithm assumes planar dynamic motion model for localization

. Prior hardware properties, sensing trends , calibrated path loss model were attempted to be incorporated in simulation. Python was selected as programming language and the code was simulated in jupyter notebook through a local host. Although functioning of this algorithm is machine independent, the device selected for running these simulation was one with Ryzen 7 4800h 2.9GHz processor chip-set.

4.4. Simulation of Probabilistic Localization Algorithm

4.4.1. Simulation Setup

The simulation operates within a grid representing the environment where the RF source is located. Grid parameters such as size and resolution are defined. A transmitter with known coordinates and specific RF propagation characteristics is placed within the grid. Parameters like reference distance, reference RSSI, path loss exponent, are set to model the RF environment. RSSI distribution across the grid is calculated based on the transmitter configuration and path loss model added to simulate real-world conditions. The simulation prompts the user to input initial coordinates to initiate the localization process.

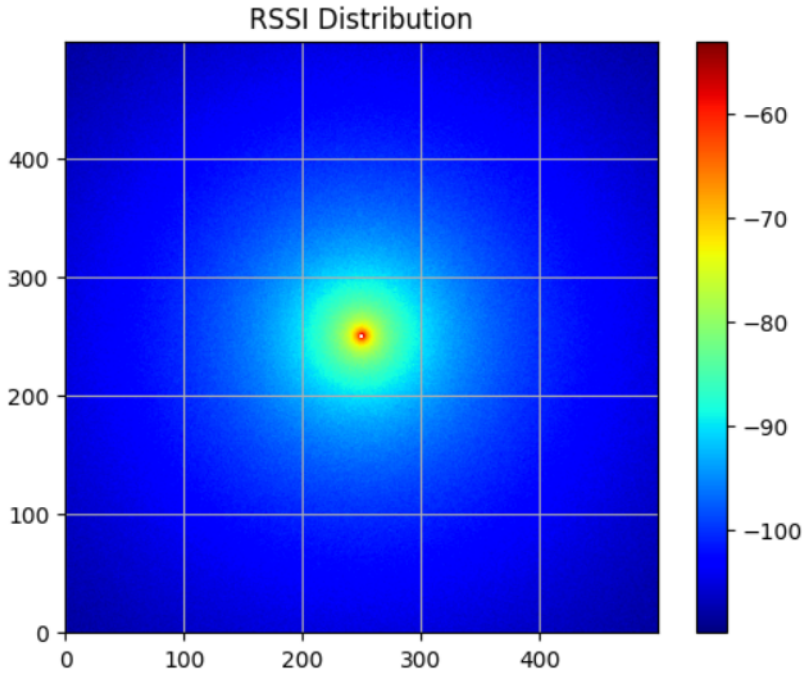

Figure 4.7: RSSI distribution over the region

4.4.2. Overview of Simulation process

- **Overview of the first case**

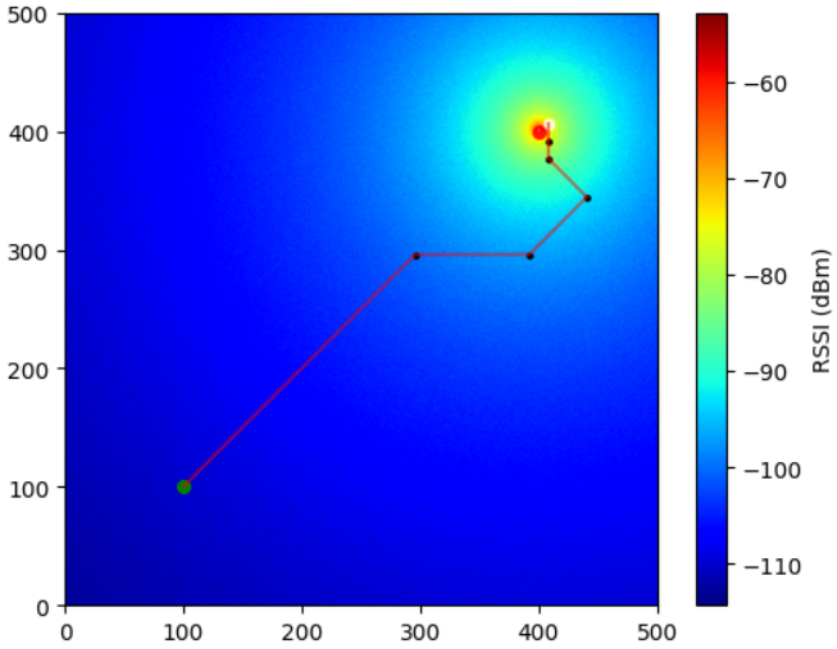

Figure 4.8: Path traced for first case

```
Enter the x-coordinate:  100
Enter the y-coordinate:  100
```

Grid Size: 500 x 500
Transmitter location: (400,400)
Receiver Location: (100,100)
Linear Distance: 424.26 meters

Final location: (402.82, 397.12)
Error: 7.83 meters
Localization achieved in 6 steps.
Total Localization time: 311.52 seconds

Analysis: The drone initally moved with correct belief however in the second iteration, its belief seems to finalize on zero degree which is off target. The drone proceeded to travel in

this direction however after this second iteration, the belief was updated in further iterations so that localization eventually converged.

- **Overview of the Long Range case**

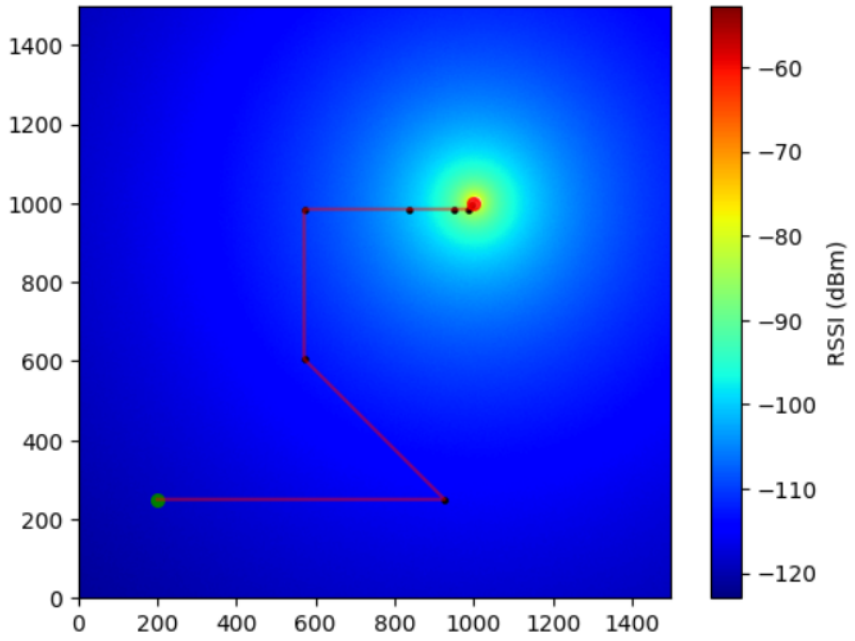

Figure 4.9: Path traced for long range case

Grid size: 1500 x 1500
Transmitter location: (1000,1000)
Receiver Location: (200,200)
Linear Distance: 1131.38 meters

Final coordinates: (998.23,998.00)
Error: 4.71 meters
Localization achieved in 8 steps.
Total Localization time: 870.83 seconds

Analysis: The drone initially carried the correct belief. However in the second iteration, it took the 135 degree heading which is off target. The drone proceeded to converge the belief by heading 90 degrees upwards followed by subsequent 0 degree heading movement in successive iterations which is the best belief here.

- **Overview of the best case**

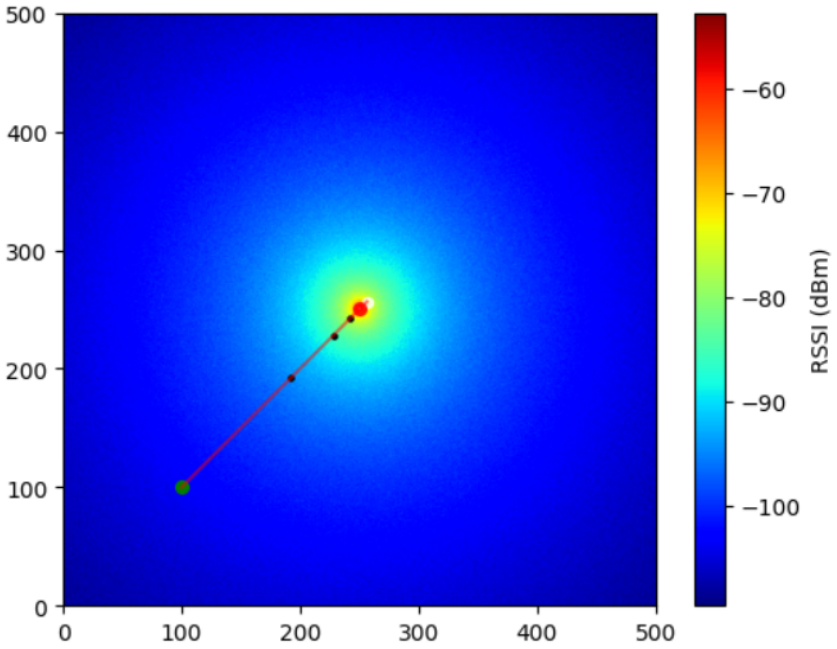

Figure 4.10: Path traced for best case

Grid size: 500 x 500
Transmitter location: (250,250)
Receiver Location: (100,100)
Linear Distance: 212.13 meters

Final coordinates: (247.47,247.47)
Error: 7.51 meters
Localization achieved in 4 steps.
Total Localization time: 163.01 seconds

Analysis: The drone carried the best initial belief and continued with best beliefs in successive iterations.

Detailed information about the parameters in each iterations can be referred to in appendix section.

4.5. Limitations

- Our localization model is applicable for stationary targets or almost stationary targets (relative to the drone) only. The model will break/fail if the target is moving
- Our model works in two dimensions only i.e. at a constant altitude.
- Our model assumes that the transmitter is emitting signal with a constant power.
- We have limited frequency range to work on.
- The testing of UAS is restricted to a controlled-open area within a visual line of sight to comply with the NCAR regulations.

4.6. Problem Faced

- Initially it was planned to purchase the commercially available antennas however since it could be purchased only in bulks of quantity, moxon antenna was fabricated manually .
- The signal was received with clear message however its RSSI value was on the lower side , showing that the received signal is weak even for shorter distances.
- The manual fabrication of antenna resulted in increased errors due to lack of precision in fabrication.
- When environmental dynamics changed, multiple new tests were required for calibration.
- Sophisticated devices such as RTL-SDR, dongles that are used for tuning, impedance matching, reflection measurements .etc were not available.
- Reflection through obstacles, multipath fading.etc affected the RSSI readings.

CHAPTER 5: CONCLUSIONS AND RECOMENDATIONS

5.1. CONCLUSIONS

An autonomous UAS developed for localization proved to be effective for real time sensing and real time path planning.

Sensing system with omni-directional and moxon antenna,where each of them receive RF signal from a single transmitter simultaneously and analyze those signals independently, was incorporated and its functionality exploited. A probabilistic localization algorithm that acted upon those RSSI data was developed.On the basis of simulations and their results, probabilistic localization algorithm involving octant filtering was proven to be capable for RF localization. The simulated results were obtained within specified error bounds, using planar dynamic model assumption and mission specific hardware parameters.

5.2. Future Enhancements

The project offers several rooms for enhancements in the future. One of the immediate action would be to incorporate double moxon antennas instead of one for better localization and to use field of view sensing approach. If accuracy becomes the major concern, and not the resources, then employing multiple UAS systems that communicate with each other in real-time, instead of just one will help develop the high fidelity localization scheme. Different filter techniques and Reinforcement Learning based decision processes such as Markov Decision Process can be used for path planning.

References

[1] L. Gupta, R. Jain, and G. Vaszkun, "Survey of important issues in uav communication networks," *IEEE communications surveys & tutorials*, vol. 18, no. 2, pp. 1123–1152, 2015.

[2] X. Zhu, Y. Feng *et al.*, "Rssi-based algorithm for indoor localization," *Communications and Network*, vol. 5, no. 02, p. 37, 2013.

[3] F. Liu, X. Cheng, D. Hua, and D. Chen, "Tpss: a time-based positioning scheme for sensor networks with short range beacons," *Wireless Sensor Networks and Applications*, pp. 175–193, 2008.

[4] S. Mohapatra, S. Kar, and S. Behera, "Different approaches of angle of arrival techniques in wireless sensor networks," *Int. J. Eng. Res. Technol.*, vol. 2, no. 2, pp. 1–9, 2013.

[5] F. Hoffmann, A. Charlish, M. Ritchie, and H. Griffiths, "Policy rollout action selection in continuous domains for sensor path planning," *IEEE Transactions on Aerospace and Electronic Systems*, vol. PP, pp. 1–1, 02 2021.

[6] H. Karl and A. Willig, *Protocols and architectures for wireless sensor networks*. John Wiley & Sons, 2007.

[7] 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*, 2001, pp. 166–179.

[8] I. Jami, M. Ali, and R. Ormondroyd, "Comparison of methods of locating and tracking cellular mobiles," 1999.

[9] S. Francis, A. Jayakar *et al.*, "Simulation based wi-fi fingerprinting for indoor localization," *Int. J. Adv. Res. Electr.*, vol. 4, no. 5, 2015.

[10] D. Niculescu and B. 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)*, vol. 3. Ieee, 2003, pp. 1734–1743.

[11] L. Yip, K. Comanor, J. C. Chen, R. E. Hudson, K. Yao, and L. Vandenberghe, "Array processing for target doa, localization, and classification based on aml and svm algorithms in sensor networks," in *Information Processing in Sensor Networks: Second International Workshop, IPSN 2003, Palo Alto, CA, USA, April 22–23, 2003 Proceedings*. Springer, 2003, pp. 269–284.

[12] J. Ash and L. Potter, "Sensor network localization via received signal strength measurements with directional antennas," in *Proceedings of the 2004 Allerton Conference on Communication, Control, and Computing*, 2004, pp. 1861–1870.

[13] L. Cong and W. Zhuang, "Hybrid tdoa/aoa mobile user location for wideband cdma cellular systems," *IEEE Transactions on Wireless Communications*, vol. 1, no. 3, pp. 439–447, 2002.

[14] G. Giorgetti, A. Cidronali, S. K. Gupta, and G. Manes, "Exploiting low-cost directional antennas in 2.4 ghz ieee 802.15. 4 wireless sensor networks," in *2007 European Conference on Wireless Technologies*. IEEE, 2007, pp. 217–220.

[15] P. Bahl and V. N. Padmanabhan, "Radar: An in-building rf-based user location and tracking system," in *Proceedings IEEE INFOCOM 2000. Conference on computer communications. Nineteenth annual joint conference of the IEEE computer and communications societies (Cat. No. 00CH37064)*, vol. 2. Ieee, 2000, pp. 775–784.

[16] T. R. Consi, J. R. Patzer, B. Moe, S. A. Bingham, and K. Rockey, "An unmanned aerial vehicle for localization of radio-tagged sturgeon: design and first test results," in *OCEANS 2015-MTS/IEEE Washington*. IEEE, 2015, pp. 1–10.

[17] J. Graefenstein, A. Albert, P. Biber, and A. Schilling, "Wireless node localization based on rssi using a rotating antenna on a mobile robot," in *2009 6Th Workshop on positioning, navigation and communication*. IEEE, 2009, pp. 253–259.

[18] Y. Chen, Z. Liu, X. Fu, B. Liu, and W. Zhao, "Theory underlying measurement of aoa with a rotating directional antenna," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 2490–2498.

[19] S. Venkateswaran, J. T. Isaacs, K. Fregene, R. Ratmansky, B. M. Sadler, J. P. Hespanha, and U. Madhow, "Rf source-seeking by a micro aerial vehicle using rotation-based angle of arrival estimates," in *2013 American Control Conference*. IEEE, 2013, pp. 2581–2587.

[20] L. K. Dressel and M. J. Kochenderfer, "Efficient and low-cost localization of radio signals with a multirotor uav," in *2018 AIAA Guidance, Navigation, and Control Conference*, 2018, p. 1845.

CHAPTER 6: Appendix

Listing 1: MATLAB Code for Polar Plotting.

```
% Replace [...] with angle data
angles = [...];

% Replace [...] with normalized RSSI data
normalized_rssi = [...];

% Convert angles to radians
angles_rad = deg2rad(angles);

% Create a polar plot
polarplot(angles_rad, normalized_rssi, 'o-');

% Set the title
title('Moxon Antenna Gain Pattern');

% Modify axes if needed
ax = gca; % Get current axes handle
ax.ThetaZeroLocation = 'top'; % Adjust theta zero
    location if needed

% Show the plot
```

Listing 2: Transmitter code in C++.

```
    #include <ESP8266WiFi.h>
#include <LoRa.h>

#define SS 15
#define RST 16
#define DIO0 2

String data = "help!";

void setup() {
  Serial.begin(9600);
  while (!Serial);

  Serial.println("Sender Host");

  LoRa.setPins(SS, RST, DIO0);
  if (!LoRa.begin(433E6)) {  // Change the frequency to
     your desired frequency (433 MHz)
    Serial.println("LoRa Error");
    while (1);
  }
}

void loop() {
  Serial.print("Sending Data: ");
  Serial.println(data);

  LoRa.beginPacket();
  LoRa.print(data);
  LoRa.endPacket();

  delay(1000);
}
```

Listing 3: Receiver code in C++.

```
include <SPI.h>
#include <LoRa.h>

#define SS_PIN 15
#define RST_PIN 16
#define DIO0_PIN 4

void setup() {
  Serial.begin(9600);
  while (!Serial);

  Serial.println("Receiver Host");

  LoRa.setPins(SS_PIN, RST_PIN, DIO0_PIN);
  if (!LoRa.begin(433E6)) {
    Serial.println("LoRa Error");
    while (1);
  }

  // Set LoRa to receive mode
  LoRa.receive();
}

void loop() {
  // Check if a packet is received
  if (LoRa.parsePacket()) {
    // Print the RSSI value
    Serial.print("RSSI: ");
    Serial.println(LoRa.packetRssi());
  }
}
```

Listing 4: Localization code in python

```
```python
from __future__ import print_function

from dronekit import connect, VehicleMode, LocationGlobal
    , LocationGlobalRelative
from pymavlink import mavutil
import time
import math
import argparse
import serial
import numpy as np

# Replace '/dev/ttyACM0' and '/dev/ttyUSB0' with the
    actual serial ports of your ESP8266 devices
ser_acm0 = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
ser_usb0 = serial.Serial('/dev/ttyUSB0', 9600, timeout=1)

parser = argparse.ArgumentParser()
parser.add_argument('--connect', default='127.0.0.1:14550
    ')
args = parser.parse_args()

# Connect to the Vehicle
print('Connecting to vehicle on: %s' % args.connect)
vehicle = connect(args.connect, baud=57600, wait_ready=
    True)

def moxon():
    def try_decode(serial_instance):
        try:
            return serial_instance.readline().decode('utf
                -8', errors='replace').strip()
        except UnicodeDecodeError:
            return "UnicodeDecodeError"

    try:
        sum_rssi = 0  # Initialize variable to store the
```
```

```
            sum of RSSI values
        num_readings = 0  # Initialize counter for number
            of readings
        while num_readings < 2:  # Read two RSSI values
            try:
                # Read a line from the serial port /dev/
                    ttyACM0
                line_ACM0 = try_decode(ser_acm0)
                # Check if the line contains the RSSI
                    value
                if line_ACM0.startswith("RSSI:"):
                    rssi_value_ACM0 = int(line_ACM0.split
                        (":")[1])
                    RSSI_M = rssi_value_ACM0
                    print("RSSI_M=", RSSI_M)
                    sum_rssi += RSSI_M
                    num_readings += 1
            except KeyboardInterrupt:
                print("Exiting...")
            break

        # Calculate the average RSSI value
        if num_readings > 0:
            average_rssi = sum_rssi / num_readings
            print("Average RSSI after 2 readings:",
                average_rssi)
            RSSIM[i] = average_rssi
    except KeyboardInterrupt:
        print("Exiting...")
    finally:
        # Close the serial port
        ser_acm0.close()

def omni():
    def try_decode(serial_instance):
        try:
            return serial_instance.readline().decode('utf
                -8', errors='replace').strip()
```

```
        except UnicodeDecodeError:
            return "UnicodeDecodeError"

    try:
        # Initialize counter for number of readings and
           variable to store sum of RSSI values
        num_readings = 0
        rssi_sum = 0

        while num_readings < 5:  # Continue looping until
            5 readings are obtained
            try:
                # Read a line from the serial port /dev/
                   ttyUSB0
                line_usb0 = try_decode(ser_usb0)
                # Check if the line contains the RSSI
                   value
                if line_usb0.startswith("RSSI:"):
                    rssi_value_usb0 = int(line_usb0.split
                       (":")[1])
                    RSSI_O = rssi_value_usb0
                    print("RSSI_0=", RSSI_O)
                    # Increment the counter after reading
                        a value
                    num_readings += 1
                    # Add the RSSI value to the sum
                    rssi_sum += RSSI_O
            except KeyboardInterrupt:
                print("Exiting...")
                break

        # Calculate the average RSSI value
        if num_readings > 0:
            RSSI0 = -53
            n=2.17
            average_rssi = rssi_sum / num_readings
            print("Average RSSI after 5 readings:",
               average_rssi)
```

```
                distance = round(10 ** ((RSSI0 - average_rssi
                    ) / (10 * n)), 2)
                print(distance)
        except KeyboardInterrupt:
            print("Exiting...")
        finally:
            # Close the serial port
            ser_usb0.close()

def arm_and_takeoff(aTargetAltitude):
    """
    Arms vehicle and fly to aTargetAltitude.
    """
    print("Basic pre-arm checks")
    # Don't let the user try to arm until autopilot is
        ready
    while not vehicle.is_armable:
        print(" Waiting for vehicle to initialise...")
        time.sleep(1)
    print("Arming motors")
    # Copter should arm in GUIDED mode
    vehicle.mode = VehicleMode("GUIDED")
    vehicle.armed = True
    while not vehicle.armed:
        print(" Waiting for arming...")
        time.sleep(1)
    print("Taking off!")
    vehicle.simple_takeoff(aTargetAltitude) # Take off to
        target altitude
    # Wait until the vehicle reaches a safe height before
        processing the goto (otherwise the command
    #  after Vehicle.simple_takeoff will execute
        immediately).
    while True:
        print(" Altitude: ", vehicle.location.
            global_relative_frame.alt)
        if vehicle.location.global_relative_frame.alt >=
            aTargetAltitude * 0.95: # Trigger just below
```

```
            target alt.
                print("Reached target altitude")
                break
        time.sleep(1)

def condition_yaw(heading, relative=False):
    if relative:
        is_relative = 1  # yaw relative to direction of
            travel
    else:
        is_relative = 0  # yaw is an absolute angle
    msg = vehicle.message_factory.command_long_encode(
        0, 0,    # target system, target component
        mavutil.mavlink.MAV_CMD_CONDITION_YAW, # command
        0, # confirmation
        heading,     # param 1, yaw in degrees
        15,          # param 2, yaw speed deg/s
        1,           # param 3, direction -1 ccw, 1 cw
        is_relative, # param 4, relative offset 1,
            absolute angle 0
        0, 0, 0)     # param 5 ~ 7 not used
    # send command to vehicle
    vehicle.send_mavlink(msg)



def send_ned_velocity(velocity_x, velocity_y, velocity_z,
    duration):
    msg = vehicle.message_factory.
        set_position_target_local_ned_encode(
        0,       # time_boot_ms (not used)
        0, 0,    # target system, target component
        mavutil.mavlink.MAV_FRAME_LOCAL_NED, # frame
        0b0000111111000111, # type_mask (only speeds
            enabled)
        0, 0, 0, # x, y, z positions (not used)
        velocity_x, velocity_y, velocity_z, # x, y, z
            velocity in m/s
```

```
            0, 0, 0, # x, y, z acceleration (not supported
                yet, ignored in GCS_Mavlink)
            0, 0)    # yaw, yaw_rate (not supported yet,
                ignored in GCS_Mavlink)
        for x in range(0, duration):
            vehicle.send_mavlink(msg)
            time.sleep(1)

    def send_global_velocity(velocity_x, velocity_y,
        velocity_z, duration):
        msg = vehicle.message_factory.
            set_position_target_global_int_encode(
            0,       # time_boot_ms (not used)
            0, 0,    # target system, target component
            mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT_INT
                , # frame
            0b0000111111000111, # type_mask (only speeds
                enabled)
            0, # lat_int - X Position in WGS84 frame in 1e7 *
                meters
            0, # lon_int - Y Position in WGS84 frame in 1e7 *
                meters
            0, # alt - Altitude in meters in AMSL altitude(
                not WGS84 if absolute or relative)
            velocity_x, # X velocity in NED frame in m/s
            velocity_y, # Y velocity in NED frame in m/s
            velocity_z, # Z velocity in NED frame in m/s
            0, 0, 0, # afx, afy, afz acceleration (not
                supported yet, ignored in GCS_Mavlink)
            0, 0)    # yaw, yaw_rate (not supported yet,
                ignored in GCS_Mavlink)
        for x in range(0, duration):
            vehicle.send_mavlink(msg)
            time.sleep(1)

    # My code begins here!

    RSSIM = [0] * 23  # Initializing RSSIM list with zeros
```

```
angle = [0] * 23

# Execute mission
# Arm and take off to an altitude of 'x' meters
arm_and_takeoff(2)

#rotate to north
print("First aligning with north!")
condition_yaw(0)
time.sleep(2)

# Read RSSI values using the moxon antenna
for i in range(23):
    angle[i] = vehicle.heading
    condition_yaw(15 * i)
    moxon()
    time.sleep(1.5)

# Convert RSSIM list to a NumPy array for easy
    manipulation
RSSIM_array = np.array(RSSIM)
angle_array  = np.array(angle)
print(RSSIM_array)
print(angle_array)

# Save RSSI and angle arrays to a data file
np.savetxt('data_file.dat', np.column_stack((RSSIM_array,
    angle_array)), fmt=['%f', '%f'])

# Find the index of the maximum RSSI value and the
    corresponding angle
max_rssi_index = np.argmax(RSSIM_array)
max_rssi_value = RSSIM_array[max_rssi_index]
max_angle = angle[max_rssi_index]

print("Maximum RSSI:", max_rssi_value)
print("Corresponding angle:", max_angle)
```

```
# Turn the drone to the corresponding angle
print("Turning the drone to the corresponding angle:",
    max_angle)
condition_yaw(max_angle)

# Now, read the RSSI value from the omni antenna
omni()

# Initiate landing
print("Setting LAND mode...")
vehicle.mode = VehicleMode("LAND")

# Close vehicle object before exiting script
print("Close vehicle object")
vehicle.close()
'''
```

Listing 5: Python Code for simulation of probabilistic method

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize
import os
from IPython.display import display, clear_output
import time

def move_point_with_waypoints(start, waypoints, target,
    duration, RSSI, run_animation=False):
    """
    Move a point from start position to target position
        passing through specified waypoints over a
        specified duration.

    Parameters:
        start (tuple): Start position of the point (x, y)
            .
        waypoints (list of tuples): List of waypoints [(
            x1, y1), (x2, y2), ...].
        target (tuple): Target position of the point (x,
            y).
        duration (float): Duration of the movement in
            seconds.
        RSSI (2D array): RSSI distribution.
        run_animation (bool): Whether to run the
            animation or not.

    Returns:
        None
    """
    # Extract start coordinates
    x_start, y_start = start

    # Initialize current position
    x_current, y_current = x_start, y_start
```

```
# Plot start position
plt.scatter(x_start, y_start, color='green', s=30)  #
    Small green point

# Plot waypoints
for waypoint in waypoints:
    plt.scatter(waypoint[0], waypoint[1], color='
       black', s=5)  # Very small black point

# Plot target position
plt.scatter(target[0], target[1], color='red', s=30,
   alpha=0.8)  # Large red point

plt.xlim(0, 500)  # Adjust as needed
plt.ylim(0, 500)  # Adjust as needed
plt.gca().set_aspect('equal', adjustable='box')  #
   Equal aspect ratio
plt.show()

# Initialize a list to store the path
path_x, path_y = [x_start], [y_start]

# Perform movement to each waypoint
for waypoint in waypoints:
    # Extract waypoint coordinates
    x_target, y_target = waypoint

    # Calculate step size for each axis
    num_steps = int(duration * 10)  # Assuming 10
       steps per second
    x_step = (x_target - x_current) / num_steps
    y_step = (y_target - y_current) / num_steps

    # Move towards the waypoint
    for _ in range(num_steps):
        # Update current position
        x_current += x_step
```

```
        y_current += y_step

        # Append current position to the path
        path_x.append(x_current)
        path_y.append(y_current)

        # Clear previous plot
        clear_output(wait=True)

        # Plot RSSI distribution
        plt.imshow(RSSI, cmap='jet', norm=Normalize()
            )
        plt.colorbar(label='RSSI (dBm)')

        # Plot path
        plt.plot(path_x, path_y, color='red', alpha
            =0.5)

        # Plot start position
        plt.scatter(x_start, y_start, color='green',
            s=30)  # Small green point

        # Plot current position
        plt.scatter(x_current, y_current, color='
            white', s=20, alpha=0.8)  # Larger white
            point

        # Plot waypoints
        for waypoint in waypoints:
            plt.scatter(waypoint[0], waypoint[1],
                color='black', s=5)  # Very small
                black point

        # Plot target position
        plt.scatter(target[0], target[1], color='red'
            , s=30, alpha=0.8)  # Large red point

        plt.xlim(0, 500)  # Adjust as needed
```

```
            plt.ylim(0, 500)  # Adjust as needed
            plt.gca().set_aspect('equal', adjustable='box
                ')  # Equal aspect ratio
            plt.show()

            # Pause for a short duration to simulate
                movement
            time.sleep(duration / num_steps)


    for _ in range(num_steps):
        # Update current position
        x_current += x_step
        y_current += y_step

        # Append current position to the path
        path_x.append(x_current)
        path_y.append(y_current)

        # Clear previous plot
        clear_output(wait=True)

        # Plot RSSI distribution
        plt.imshow(RSSI, cmap='jet', norm=Normalize())
        plt.colorbar(label='RSSI (dBm)')

        # Plot path
        plt.plot(path_x, path_y, color='red', alpha=0.5)

        # Plot start position
        plt.scatter(x_start, y_start, color='green', s
            =30)  # Small green point

        # Plot current position
        plt.scatter(x_current, y_current, color='white',
            s=20, alpha=0.8)  # Larger white point

        # Plot waypoints
```

```
            for waypoint in waypoints:
                plt.scatter(waypoint[0], waypoint[1], color='
                    black', s=5)  # Very small black point

            # Plot target position
            plt.scatter(target[0], target[1], color='red', s
                =30, alpha=0.8)  # Large red point

            plt.xlim(0, 500)  # Adjust as needed
            plt.ylim(0, 500)  # Adjust as needed
            plt.gca().set_aspect('equal', adjustable='box')
                # Equal aspect ratio
            plt.show()

            # Pause for a short duration to simulate movement
            time.sleep(duration / num_steps)

# Define grid size and grid points
grid_size = 500
step_size = 1
X, Y = np.meshgrid(np.arange(0, grid_size, step_size), np
    .arange(0, grid_size, step_size))

# Define transmitter location
tx_x, tx_y = 400,400

# Define reference distance and reference RSSI
d0, RSSI0 = 1, -53

# Define path loss exponent
n = 2.17

# Define variance of the Gaussian noise
sigma = 2
# Calculate distance of each grid point from the
    transmitter
D = np.sqrt((X - tx_x)**2 + (Y - tx_y)**2)
```

```
# Calculate path loss
path_loss = 10 * n * np.log10(D / d0)

# Generate AWGN noise for each grid point separately
AWGN = np.random.randn(grid_size, grid_size)
AWGN_mean = AWGN * sigma

# Calculate the RSSI
RSSI = RSSI0 - path_loss - AWGN_mean

# Plot the RSSI values
fig, ax = plt.subplots()
im = ax.imshow(RSSI, cmap='jet', norm=Normalize())
cbar = fig.colorbar(im, ax=ax)
#ax.set_xlabel('X', fontweight='bold')
#ax.set_ylabel('Y', fontweight='bold')
ax.invert_yaxis()
ax.set_aspect('equal', adjustable='box')
plt.title('RSSI Distribution')
plt.grid(True)
plt.show()

# Prompt the user to input a coordinate
x_init = float(input("Enter the x-coordinate: "))
y_init = float(input("Enter the y-coordinate: "))

# Initialize a list to store all coordinates
all_coordinates = []

# Append target coordinates to the list
all_coordinates.append([tx_x, tx_y])

# Append user input coordinates to the list
all_coordinates.append([x_init, y_init])

print("\n\n")

# Initialize the iteration count
```

```
p = 0

# Initialize the distance threshold
dost = 500
sum=0
while dost > 20:
    # Increment iteration count
    p += 1
    print(f"In Iteration {p}")

    # Calculate the RSSI value at the desired point
    d = np.sqrt((x_init - tx_x)**2 + (y_init - tx_y)**2)
    pathloss = 10 * n * np.log10(d / d0)
    RSSII = RSSI0 - pathloss - AWGN_mean[int(x_init), int
        (y_init)]
    print(f"The RSSI value at the point ({x_init:.2f}, {
        y_init:.2f}) is {RSSII:.2f} dBm")

        print("\n")

    # Calculate the RSSI at each coordinate and store it
        in an array
    R_octants = np.zeros(len(coords))
    for i in range(len(coords)):
        d = np.sqrt((coords[i][0] - tx_x)**2 + (coords[i
            ][1] - tx_y)**2)
        pathloss = 10 * n * np.log10(d / d0)
        R_octants[i] = RSSI0 - pathloss - AWGN_mean[int(
            coords[i][0]), int(coords[i][1])]

    # Print the coordinates and their corresponding RSSI
        values
    print("Coordinate          RSSI (dBm)")
    print("----------------  ----------")
    for i in range(len(coords)):
        print(f"({coords[i][0]:.2f}, {coords[i][1]:.2f})
            {R_octants[i]:.2f}")
```

```
# Find the maximum RSSI value
max_RSSI = np.max(R_octants)

# Find the index of the maximum RSSI value
max_index = np.argmax(R_octants)

print("\n")
# Print the maximum rssi value and index of the
    maximum RSSI value
print(f"The maximum RSSI value is {max_RSSI:.2f}dBm")
print(f"The index of the maximum RSSI value is {
    max_index}")

# Calculate the bearing corresponding to the index
bearing = max_index * (1/8) * 360 % 360
print(f"The bearing is {bearing:.2f} degrees")

# calculating distances
dist = round(10 ** ((RSSI0 - RSSII) / (10 * n)), 2)
ed = dist * 0.95
sum+= ed

# Print the estimated distance and iterative distance
print(f"The estimated distance is {dist:.2f} meters")
print(f"The iterative distance is {ed:.2f} meters")

# Calculate the updated (x, y) values
x_updated = x_init + ed * np.cos(np.deg2rad(bearing))
y_updated = y_init + ed * np.sin(np.deg2rad(bearing))

# Store the updated coordinates in the array
all_coordinates.append([x_updated, y_updated])


# Store the updated (x, y) values in a tuple
updated_coords = (x_updated, y_updated)

# Print the updated coordinates
```

```
        print(f"The updated coordinates are ({x_updated:.2f},
            {y_updated:.2f})")


        # Update x_init and y_init for the next iteration
        x_init, y_init = x_updated, y_updated

        # Update dost for the loop condition
        dost = dist
        print("\n\n")


#print(f"Total travel distance {sum-ed} meters")

# Display the array of coordinates
print(f"Array of coordinates for localization:")
for idx, coordinates in enumerate(all_coordinates):
        print(f"Point {idx}: ({coordinates[0]:.2f}, {
            coordinates[1]:.2f})")



print("\n\n")

drone_speed = 3

# Print the number of iterations
print(f"Localisation achieved in {p} steps")

print("\n\n")


#Localization Time Calculation
l_t= (sum-ed)/(drone_speed) + p*24
print(f"The total localization time is {l_t:.2f} seconds"
    )
print("\n\n")
```

```
# Prompt user if they want to run the animation
run_animation_input = input("Do you want to run the
    animation? (y/n): ")

if run_animation_input.lower() == 'y':
    # Add a delay of 5 seconds
    time.sleep(3)

    # Create a new figure for the animation
    plt.figure()

    # Move point with waypoints
    start_position = all_coordinates[1]  # Start position
    waypoints = all_coordinates[2:-1]  # Waypoints
    target_position = all_coordinates[0]  # Target
        position
    movement_duration = 0.3 # in seconds

    move_point_with_waypoints(start_position, waypoints,
        target_position, movement_duration, RSSI,
        run_animation=True)
else:
    print("Animation skipped.")
```

Listing 6: Python Code for probabilistic method

```
  import os
import numpy as np
import time

def read_dat_file(file_path):
    bearing_array = []
    moxon_readings_array = []
    omni_readings_array = []

    try:
        with open(file_path, 'r') as file:
            for line in file:
                columns = line.strip().split()  #
                    assuming columns are separated by
                    whitespace
                if len(columns) >= 3:  # make sure there
                    are at least three columns
                    if columns[0] != 'Mean_RSSI':  # skip
                        mean value line
                        bearing_array.append(float(
                            columns[0]))
                        moxon_readings_array.append(float
                            (columns[1]))
                        omni_readings_array.append(float(
                            columns[2]))
                else:
                    print(":::", line)

    except FileNotFoundError:
        print("File not found.")
    except Exception as e:
        print("An error occurred while reading the file:"
            , e)

    return bearing_array, moxon_readings_array,
        omni_readings_array
```

```
def write_mean_to_dat(file_path, mean_rssi):
    try:
        with open(file_path, 'a') as file:
            file.write(f"Mean_RSSI\t\t{mean_rssi:.2f}\n")
        print("Mean RSSI appended to file.")

    except IOError:
        print("Error writing mean RSSI to file.")

def calculate_distance(RSSI, RSSI0, n):
    distance = round(10 ** ((RSSI0 - RSSI) / (10 * n)),
       2)
    return distance

# Example values for RSSI0 and n
RSSI0 = -53  # Example RSSI0 value
n = 2.17  # Example path loss index
velocity = 3  # m/s
t_factor = 0.65

# Example usage:
current_dir = os.path.dirname(os.path.abspath(__file__))
file_name = 'data_file.dat'
file_path = os.path.join(current_dir, file_name)

bearing_array, moxon_readings_array, omni_readings_array
   = read_dat_file(file_path)

mean_rssi = round(sum(omni_readings_array) / len(
   omni_readings_array), 2)

write_mean_to_dat(file_path, mean_rssi)

# Calculate distance using mean_rssi
distance = calculate_distance(mean_rssi, RSSI0, n)
print("Distance:", distance, "meters")

# Calculate predicted time
```

```
predicted_time = round(distance / velocity, 2)
print("Predicted Time:", predicted_time, "seconds")

# Calculate time_step
time_step = round(t_factor * predicted_time, 2)
print("Time factor:", round(t_factor, 2))
print("Time Step:", time_step, "seconds")

# Process moxon_readings_array
octants_value = [moxon_readings_array[i] for i in range
    (0, len(moxon_readings_array) - 1, 3)]  # Skip the 26
    th value
path_belief = [0.125] * 8  # Initialize path belief with
    8 elements each with value 0.125

# Create relevant_readings array
relevant_readings = octants_value

# Normalize relevant_readings array between 0 and 1
min_reading = min(relevant_readings)
max_reading = max(relevant_readings)
normalized_readings = [(value - min_reading) / (
    max_reading - min_reading) for value in
    relevant_readings]

# Find indexes of top three values in normalized_readings
top_indexes = sorted(range(len(normalized_readings)), key
    =lambda i: normalized_readings[i], reverse=True)[:3]
a, b, c = top_indexes

# Print octants array
print("\nOctants Array:")
print(octants_value)

# Print relevant_readings array
print("\nRelevant Readings Array (Normalized):")
print(normalized_readings)
```

```
# Print top indexes
print("\nTop Indexes (a, b, c):", a, b, c)

# Compute selection criteria
if (b == a - 1) or (b == a + 1) or (a == 0 and b == 7):
    print("Take :", a)
else:
    print("Remeasurement ")

# Check distance criteria
if distance < 20:
    print("Localization achieved within set error limits"
        )

# Append relevant parameters to .dat file
try:
    with open(file_path, 'a') as file:
        file.write(f"Distance\t\t{distance:.2f} meters\n"
            )
        file.write(f"Time factor\t\t{t_factor:.2f} \n")
        file.write(f"Predicted_Time\t{predicted_time:.2f}
            seconds\n")
        file.write(f"Time_Step\t{time_step:.2f} seconds\n
            ")
        file.write("Octants_Array\t" + ', '.join(map(str,
             octants_value)) + '\n')
        file.write("Relevant_Readings_Array_Normalized\t"
             + ', '.join(map(str, normalized_readings)) +
            '\n')
        file.write(f"Top_Indexes\ta: {a}, b: {b}, c: {c}\
            n")
        if (b == a - 1) or (b == a + 1) or (a == 0 and b
            == 7):
            file.write("Selection_Criteria\tTake a\n")
        else:
            file.write("Selection_Criteria\tRemeasurement
                \n")
        if distance < 20:
```

```
                file.write("Localization Achieved\
                    tLocalization achieved within set error
                    limits\n")
except IOError:
    print("Error writing parameters to file.")

# Rename current file with timestamp as new filename
current_time = time.strftime("%Y-%m-%d_%H-%M-%S")
new_file_name = f"data_file_{current_time}.dat"
new_file_path = os.path.join(current_dir, new_file_name)
os.rename(file_path, new_file_path)

print(f"File renamed to: {new_file_name}")
```

Listing 7: Python Code for probabilistic method

```
import numpy as np
import matplotlib.pyplot as plt

# Function to generate random RSSI values within the
    given limits
def generate_random_RSSI_values(lower_limit, upper_limit,
    size):
    return np.random.uniform(lower_limit, upper_limit,
        size)

# Function to normalize RSSI values
def normalize_rssi_values(RSSI_values):
    min_RSSI = min(RSSI_values)
    max_RSSI = max(RSSI_values)
    RSSI_values_normalized = (RSSI_values - min_RSSI) / (
        max_RSSI - min_RSSI)
    return RSSI_values_normalized

# Function to plot RSSI values on a polar plot
def plot_rssi_values_polar(RSSI_values_normalized):
    plt.figure(figsize=(8, 8))
    plt.subplot(projection='polar')
    plt.plot(np.deg2rad(range(0, 360, 15)),
```

```
        RSSI_values_normalized)
    plt.title('RSSI Values on Polar Plot')
    plt.show()

# Function to find peaks at octants
def find_peaks_at_octants(RSSI_values_normalized):
    octant_peaks = [[] for _ in range(8)]
    for bearing in range(0, 360, 45):
        octant_index = bearing // 45
        octant_peaks[octant_index].append((bearing,
            RSSI_values_normalized[bearing // 15]))
    return octant_peaks

# Function to plot top 3 peaks before normalization
def plot_top_3_peaks_before_normalization(top_peaks):
    fig, ax = plt.subplots(subplot_kw={'projection': '
        polar'}, figsize=(8, 8))
    for i, (bearing, value) in enumerate(top_peaks, 1):
        ax.plot(np.deg2rad(bearing), value, 'o',
            markersize=8, label=f'Peak {i} (Before
            Normalization)')
    ax.set_title("Top 3 Peaks (Before Normalization)")
    ax.legend()
    plt.show()

# Function to create an array of size 8
def create_array_1x8(bearing_values,
    normalized_peak_values):
    array_1x8 = np.zeros(8)
    for bearing, value in zip(bearing_values,
        normalized_peak_values):
        array_1x8[bearing // 45] = value
    return array_1x8

# Loop for 10 iterations
for i in range(1, 11):
    print(f"Iteration {i}:")
```

```
# Generate random RSSI values with increasing limits
lower_limit = -130 + (i - 1) * 10
upper_limit = -120 + (i - 1) * 10
RSSI_values = generate_random_RSSI_values(lower_limit
    , upper_limit, 24)

# Print the raw data array
print("Raw Data Array:", RSSI_values)

# Normalize RSSI values
RSSI_values_normalized = normalize_rssi_values(
    RSSI_values)

# Plot RSSI values on a polar plot
plot_rssi_values_polar(RSSI_values_normalized)

# Find the top 3 peaks before normalization
octant_peaks = find_peaks_at_octants(
    RSSI_values_normalized)
all_peaks = [peak for octant in octant_peaks for peak
     in octant]
top_3_peaks_before_normalization = sorted(all_peaks,
    key=lambda x: x[1], reverse=True)[:3]

# Display the top three peaks along with their
    bearings
print("Top 3 Peaks (Before Normalization):")
for j, (bearing, value) in enumerate(
    top_3_peaks_before_normalization, 1):
      print(f"{j}. Bearing: {bearing}, RSSI Value: {
          value}")

# Plot the top 3 peaks before normalization
plot_top_3_peaks_before_normalization(
    top_3_peaks_before_normalization)

# Divide each peak by the smallest peak value
smallest_peak_value = min(
```

```
    top_3_peaks_before_normalization, key=lambda x: x
    [1])[1]
normalized_peak_values = [peak[1] /
    smallest_peak_value for peak in
    top_3_peaks_before_normalization]

# Assign normalized peak values to the positions
    corresponding to their respective bearing angles
bearing_values = [peak[0] for peak in
    top_3_peaks_before_normalization]
array_1x8 = create_array_1x8(bearing_values,
    normalized_peak_values)

# Print the formatted array
print("\n1x8 Array with Normalized Peak Values:")
print(np.array2string(array_1x8, separator=', ',
    precision=4, suppress_small=True))

# Sum of all elements in the array
array_sum = np.sum(array_1x8)

# Normalize the array by dividing each element by the
    sum of all elements
normalized_array = array_1x8 / array_sum

# Print the normalized array
print("Normalized array:", normalized_array)

# Find the index of the largest element
step_direction = np.argmax(normalized_array)

# Print the index of the largest element as
    step_direction
print("Index of the largest element:", step_direction
    )
print("\n" + "-"*50 + "\n")
```