Path Detection to Acoustic Source in Dynamic Environment

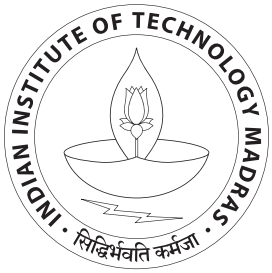
A Project Report

submitted by

RISHIKA VARMA K

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.
May 2022

THESIS CERTIFICATE

This is to certify that the thesis entitled **Path Detection to Acoustic Source in Dynamic Environment**, submitted by **Rishika Varma K** (**CS18B045**), to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelors of Technology**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr.Ayon Chakraborty
Research Guide
Assistant Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 27.05.2022

ACKNOWLEDGEMENTS

I would like to thank Professor Ayon for being a wonderful guide for this project along my UGRC as well as BTP journey. It was very fulfilling to work under him and I learnt a lot about the sensing field from him.Through the progress of this project, we explored various aspects of problem-solving. Although from the surface, the problem may not appear to pose an issue on further investigation, there was more to it than was evident. Thus, trying to implement our ideas, testing them to see the problems posed, and then figuring out where the issue arose to find new and innovative ways to fix it after discussion helped me learn a lot in ways I did not expect. I learned to be patient and not jump to conclusions but rather analyze and authenticate my suspicions and then think of ways to improve my solution. I also learned that research is not entirely about reaching the result but instead savoring each step in finding out exactly why something must be done a certain way instead of other possibilities. This helped me broaden my perspective and push on to find better answers even after reaching a feasible solution that may not be the best suited in the situation.

ABSTRACT

KEYWORDS: Sound Source, Localization, Sensing, Graph Creation, Graph Search,

In this project, we sought to explore the localization of a sound source using modalities like intensity and angle of arrival of those sound rays to the receiver in a dynamic environment where the source is not directly in line of sight. To achieve this we structured the problem into graph creation and graph search. For the graph creation we used clustering techniques on data obtained from a LiDAR which gives the immediate point cloud to make children nodes and thus incrementally create a graph as the receiver traverses through it. The second part of problem is the effective search of the graph that we have created and for this we are using a microphone array to detect the angle of arrival of the sound. The point cloud data when maintained helps create the floor plan of the environment and this helps to retrace sound rays using angle of arrival and gauge the direction from which sound is emitted. We also use direction of maximum intensity of the sound to give another metric for the graph search. Using these criteria we tried to come with a system that incrementally improves its search while moving towards the source of the sound. This system is effective for situations like public safety scenarios where prior knowledge of layout and other details are unlikely to be available.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

AoA	Angle of Arrival
DFS	Depth first search
IMU	Inertial Measurement Unit
LiDAR	Light Detection and Ranging

CHAPTER 1

INTRODUCTION

This project explores how to localize a source that may not be in the line of sight using alternate methods like sound. We try to use data obtained from the sound signal emitted by the source and use it along with the contour of the room to get an idea regarding the direction towards the source. The contour of the room could also be obtained similarly using acoustic signals by emitting specific signals from the receiver and analyzing the reflected versions of these signals detected by the sensors. The idea is to use the angle of arrival Barabell [1983]; Schmidt [1986] as detected by the sensor and the room geometry to get an accurate direction towards the source. An alternative to this would be to use the intensity of sound as a guide of the direction to be explored, but this could be faulty, and it is only a way of improving from a completely blind search. For this, the signal of the contour of the room that was obtained must be converted into a graph where the nodes are the points to which the receiver can move. We must then analyze the nodes using the acoustic signal information from the source and decide on the correct or (at least) the most optimal node towards which the receiver must move so that path travelled in reaching the source is minimized. This project is helpful in many scenarios. One of the main applications is to help in public safety situations. In the current case, first responders like firefighters are forced to approach a dangerous situation like a fire or flood without much help from technology. Since the internet is likely to be compromised in such cases, it is not easy to know the layout of the area or other such details. Thus any solution to help them must involve a device

that works independently and analyses the relatively safe and quick path towards people who need to be rescued. Visibility in these situations may also be quite compromised due to smoke or movement, so using acoustic signals is a better solution. Using cameras instead of sound signals for layout may also result in compromise of privacy concerns. Another scenario where this could be beneficial is for devices like Alexa. Having an idea of the location from where the command is given could improve the context and, consequently, the device's effectiveness. For example, for an instruction to switch on the light, the device must know which room is being referred to, which could be obtained from the speaker's location.

CHAPTER 2

MOTIVATION AND CHALLENGES

The source may be in the receiver's line of sight, or otherwise when it is in line of sight applications involving a camera that can detect it. One application that uses sound to localise the source is the Voloc Shen *et al.* [2020] which uses the idea that the receiver is likely to be located near a wall and uses AoA of the first two rays to figure out the distance to the wall as well as the speaker. This does not work for us as the receiver should be able to move in our case and need not be near a wall. When the source is in line of sight, a similar approach could be used where AoA is used to localise the source, but dependence on the wall is avoided. In the case of non-line of sight scenarios, a more rigorous solution involving the AoA of sound is required. Directly going towards the direction of initial AoA will not serve our cause as there might be no rays directly coming from the source. All rays might be reflected, and the direction pointed to by initial rays may not be the most optimal. Another paper that deals with a similar issue and uses an approach closer to our idea is Multiresolution Analysis of Voice Localization Pradhan *et al.* [2018]; Wang *et al.* [2021]. This approach uses the initial rays and the room contour to retrace the path of these rays and find the location of the source. While this method is convenient to find the exact location of the source, it is only easy to figure this out when the floor plan is not too complicated, and the receiver could detect the whole plan until the source. It is also challenging to know the path towards the source as the floor plan may be complicated. We plan to take a different approach where our algorithm searches the space as a graph. This can be done by making a graph

where each node represents the directions in which the receiver can travel along with distance. Searching this graph blindly by exploring all children until they have been exhausted can take a very long time. Thus we devise a cost function to help us explore the graph effectively. This cost function gives an idea of what node to explore to reach the destination as quickly as possible. Some factors that could be used are sound intensity, AoA and edge lengths. A combination of all these factors could also give an efficient solution. Thus the crux of this problem comes down to two steps: i) developing an algorithm that creates a graph from the point cloud detected by the receiver and ii) creating a cost function using signals from the source that helps explore the created graph effectively.

CHAPTER 3

SYSTEM SETUP

3.1 Simulation setup

3.1.1 Floor Plan

Before moving to the practical aspect of this setup, we have used a simulation of the setup. The first step for this is to create the environment. This was done by using a matrix to represent the floor plan. Each index of the matrix represents a small area of the existing floor. If there is some obstacle or wall at that point, the matrix value will be one, and if the receiver can travel to that point, the value would be zero. Depending on the nature of the obstacle, we gave angle values for each of the points that are obstacles based on the angle of the surface of the obstacle it is a part of. In this matrix, two positions are randomly selected to represent the source and receiver positions. An example of a floor plan that was generated along with source and receiver locations is shown in figure 3.1. We follow this example through all the following algorithms.

3.1.2 Ray Tracing

Now to simulate the acoustic signals, we started tracing the paths of sound rays emerging from the source in all directions. The path traveled by the ray is followed until it has reached the maximum distance at which it attenuates (This threshold

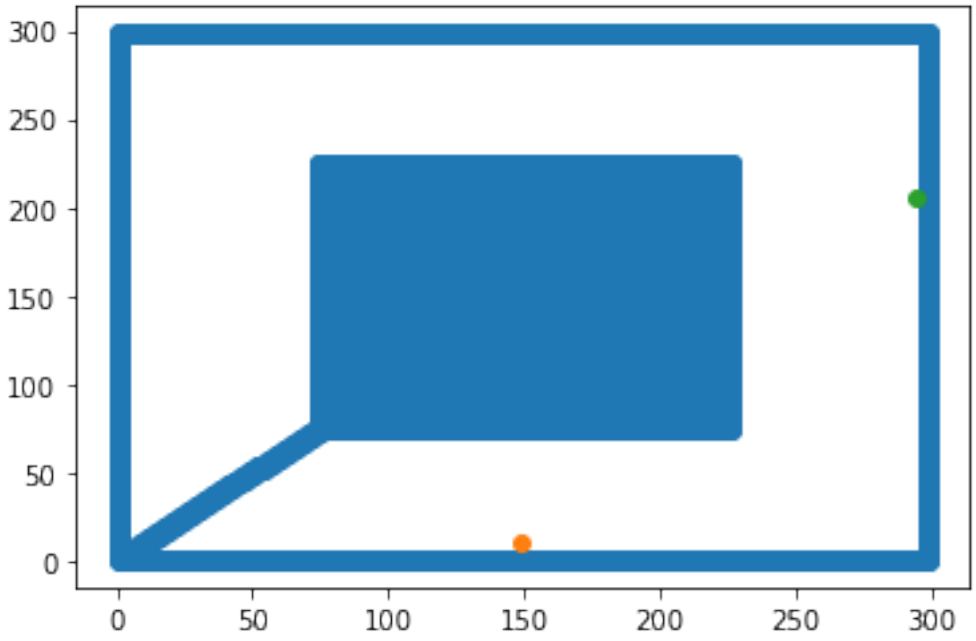


Figure 3.1: An example of the generated floor plan where orange dot is the source and green dot is the receiver.

is also given by us). Any time one of the rays reaches an obstacle, it changes its direction (reflects) appropriate to the angle of incidence and that of the wall.Figure 3.2 shows the trajectory of three of the paths starting from the source until they attenuate for the source location and floor plan in figure 1. We add intensity corresponding to each ray (inversely proportional to the distance from source by a factor α) at all points that it reaches. We also decrease the intensity by a factor(β) during these reflections. The pseudo code of the algorithm for tracing the path of the ray and giving appropriate intensity and Angle of Arrival (AoA) values is given by Algorithm 1. Both of these quantities are taken to be the average and intensity weighted average of respective corresponding values of all the rays that reach that point. Some amount of error is also added to the angle of arrival calculation to account for variations while reflecting. While this algorithm allows for transmission, reflection and attenuation, due to the discrete point nature of the simulation where a small area of the floor is represented by one number, which

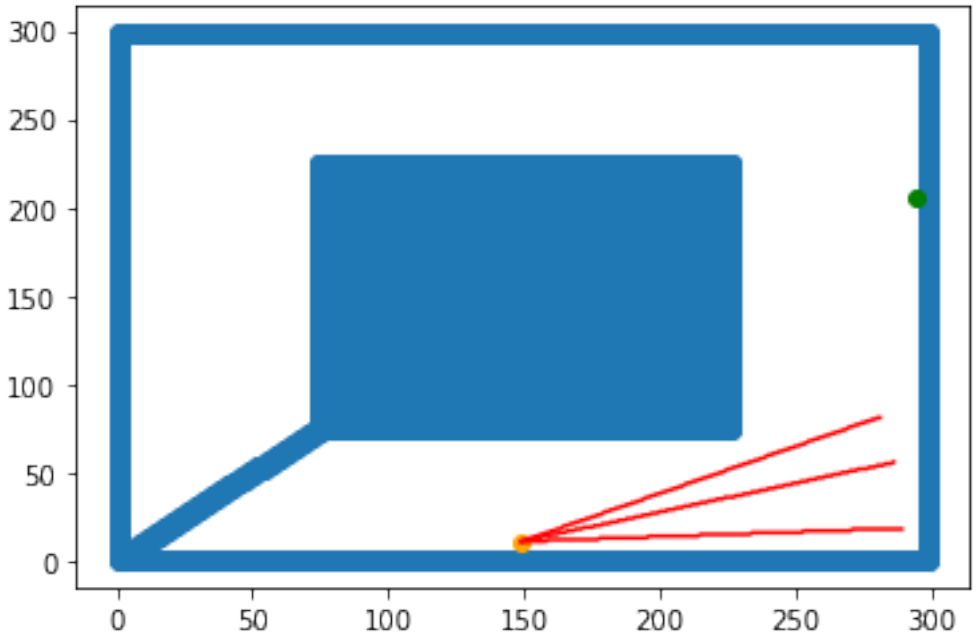


Figure 3.2: Showing some of the traced rays

results in the simulation resulting in the implication of small gaps between these blocks. Due to this, the unwanted consequence is internal reflection between these discrete blocks however, in reality there is no gap between them. To prevent this constraint was added to make sure that reflection did not happen between blocks and a ray can only reflect if it passes through some empty blocks that aren't solid before reaching the reflection block. Put differently, a ray cannot reflect between contiguous blocks.

3.1.3 Point Cloud

The next step of the simulation was to find the point cloud information of the receiver. For this, we used the same principle as the one for tracing the rays but modified it slightly. In each direction, we found the first point reached by the ray traced that is an obstacle. Depending on whether this is within the threshold of the maximum distance detectable (given by us), the point is marked as part of

Algorithm 1 RAY DFS

```
procedure RAY_DFS(source, angle, inten_fac, dist)
    if dist >= distance_limit then
        return
    end_point ← point in the direction of ray at the edge of the matrix
    all_points ← all points between source and end_point    ▷ Using bresenham library
    while p in points do
        if prev_point_flag == 1 then    ▷ prev_point_flag indicates whether the previous point is a flag
            if floor_plan[p]! = 0 then
                continue
            else prev_point_flag ← 0
        intense ← inten_fac/dist(source, p) ∗ α
        AoA[p] ← (intensity[p]∗AoA[p]+intense∗angle)/(intense+intensity[p])+ϵ
        intensity[p] ← intensity[p] + intense
        if floor_plan[p]! = 0 then
            prev_point_flag ← 1
            ray_dfs(p, 2 ∗ angle[p] − angle, intense/β, dist + dist(source, p))
            break
```

the point cloud in that direction or left empty. A certain amount of error is added to depict ground reality. Figure 3.3 gives the point cloud(red points) generated by the algorithm for the initial receiver position for the setup in Figure 1. It is plotted along with the floor plan(blue points). The pseudo code for the algorithm for getting this point cloud is shown by Algorithm 2.

3.2 Physical setup

The physical setup consists of three components. The first component involves the LIDAR which will give the point cloud information to the algorithm. The second component of the setup would be the IMU and the last component is the microphone array. These components are connected using a raspberry pi. Using this setup, the graph can be created, and the cost of each node is calculated based

Algorithm 2 Point Cloud

```
1: procedure POINT_CLOUD(receiver)
2:     angle ← 0
3:     while angle < 360 do
4:         end_point ← point in the direction of ray at the edge of the matrix
5:         all_points ← all points between source and end_point ▷ Using
   bresenham library
6:         while p in points do
7:             if floor_plan[p]! = 0 then
8:                 if dist(receiver, p) > max_dist then
9:                     contour[angle] = None
10:                else contour[angle] = p + ϵ ▷ ϵ gives error
11:                break
12:        i ← i + 1
13:    return contour
```

on which area is searched.

3.2.1 LiDAR

The LiDAR is a sensor that gives a point cloud of the obstacles surrounding the user. The figure 3.4 shows an image of a LiDAR. It emits sound rays in the form of chirps in various directions which then reflect and are detected by a receiver after a certain interval of time. This interval helps gauge the amount of distance at which a point of the obstacle at which the ray reflected is situated from the location of the user. By moving with the LiDAR continuously the user can get an estimate of the obstacles and walls in his vicinity. While this is essential for the generation of graph in our problem to be able to search it and reach the source, it also helps to aid search. The first purpose is to enable AoA aided search. Through the microphone array (as discussed following) we get the angle of arrival information. For the graph creation we do not need to maintain previous point cloud information however, to be able to use AoA we should be able to retrace the ray to towards the source,

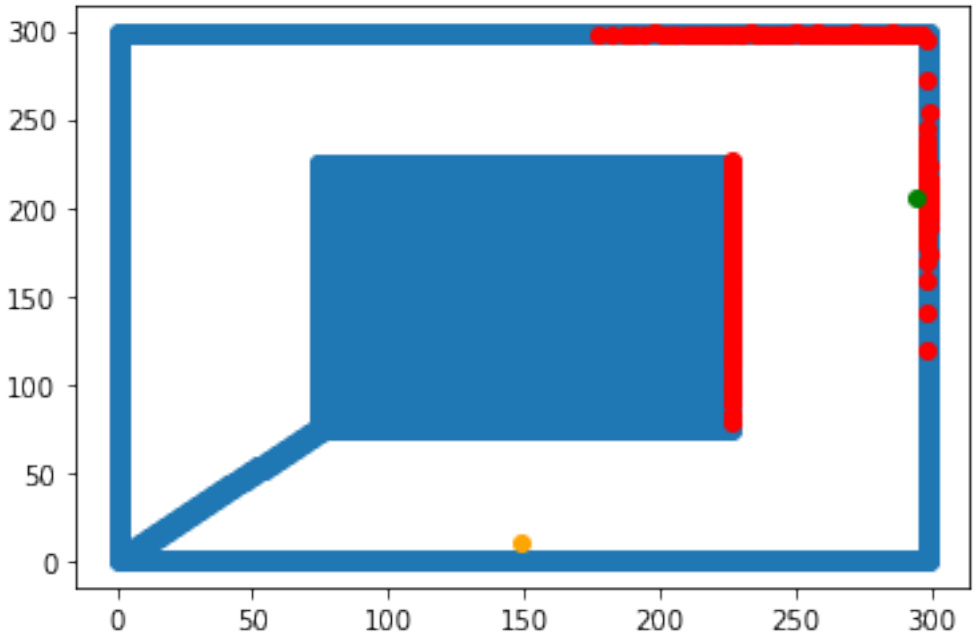


Figure 3.3: Showing the point cloud generated

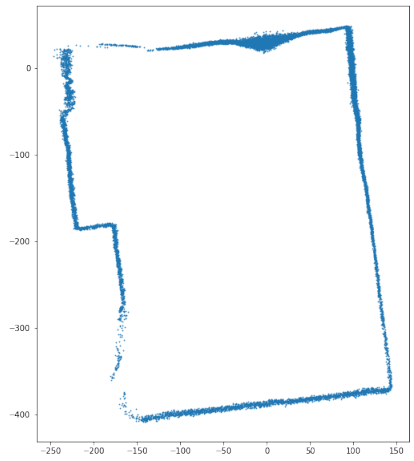
to achieve this by maintaining point cloud information of previous states, we can incrementally create the floor map of the place. Figure 3.5 shows the point cloud data of a room from different points. Initially due to low information only a part of it is known but as the user moves more information will allow the creation of better floor map. Using this map, we can retrace the path taken by the sound rays from the source to the receiver and along with appropriate reflections thus estimate the direction from which the sound rays are emerging which gives an estimation of the direction of the source. This can aid in the heuristic function to determine the search direction along with sound intensity. To be able to achieve this one algorithm that can be used is the simultaneous Localization and mapping (SLAM) Dissanayake *et al.* [2001] algorithm. This algorithm takes the point cloud information at each point and simultaneously maps them to that already existing along with mapping the path followed by the receiver. This is done by shape matching the point cloud information at various points and consequently this gives an indication of the path followed by the user as well. As the core SLAM

Figure 3.4: 2D LiDAR sensors

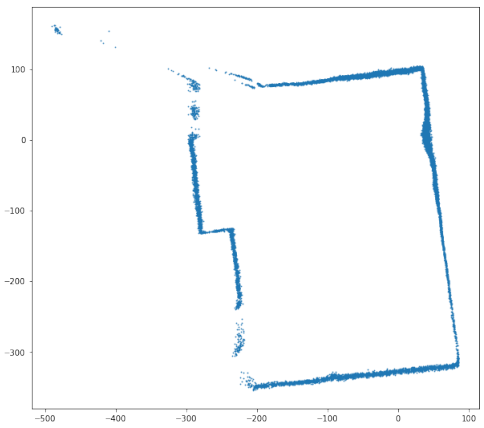
algorithm which requires the LiDAR to roam through the entire room gives the path followed as well, here in our case an adaptation of it that doesn't allow for randomised movement but uses whatever information is obtained from the path traversed to give some idea can be used. The path traversed although not accurate can help aid the IMU information to give a more accurate position of the receiver as well. There also exists an open point cloud library with various algorithms for clustering, recognition, filters, etc,. It is a library for 3D point clouds but in our case at the moment we only have 2D point cloud in the plane at which the LiDAR is placed. We consider the walls as only lines as we do not need more than the floor plan. In future our algorithm could be adapted into taking 3D data in consideration as this is necessary in cases where the obstacles are not of full height. But even in the 2D case the algorithms in the PCL can be adapted to our situation.

3.2.2 IMU

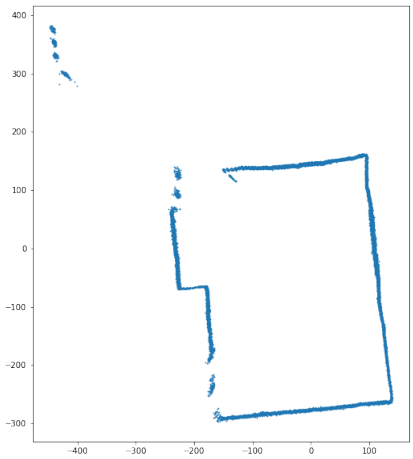
The inertial measurement unit (IMU) is a device that consists of a accelerometer, magnetometer and gyroscope that helps the user to track the path traversed. Fig-



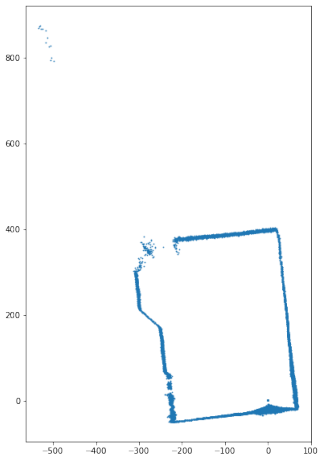
(a) Point cloud from point 1



(b) Point cloud from point 2



(c) Point cloud from point 3



(d) Point cloud from point 4

Figure 3.5: Point cloud plots from various positions in the same environment

ure 3.6 shows an image of the same. The accelerometer gives linear acceleration on an axes calibrated from the perspective of the IMU. By double integrating this data discretely over appropriate limits taking directions into consideration we can get a rough idea of the linear path taken by the IMU. The gyroscope gives the angular acceleration of the IMU as measured from its axes. This will give the direction changes experienced by the user. The third component, the magnetometer gives information about the orientation of the IMU which allows us to understand the change in direction of the axes of observation from which the linear and angular accelerations are measured. All the information used together helps give the path

Figure 3.6: IMU sensors

taken by the user holding an IMU. There are also algorithms that take strides into consideration instead of linear acceleration in x or y direction and use it to measure distance covered. The number of strides can be gauged by peaks in the linear acceleration of the z direction. All this data mutually enhances the available information to reduce error thus allowing for a more accurate result. One such algorithm that implements this approach is the Pedestrian Dead Reckoning (PDR) Liu *et al.* [2016] algorithm which takes each data point and calculates the new position from there assuming it to be continuous until the next data point. The pseudo code for this algorithm is shown by algorithm 3. For filtering error form this data we can use a Kalman filter Li *et al.* [2015] which assumes that the error is a distributed normally. This is not entirely accurate in this case, and to rectify that we can use the extended Kalman filter which does not take this assumption. This implementation of this algorithm is in progress as it has some issues regarding taking device orientation into account in case of the angular acceleration. I have also tried use an existing implementation of the PDR algorithm but this was not entirely accurate either. The semi accurate path plotted by this algorithm is shown by figure 3.7.

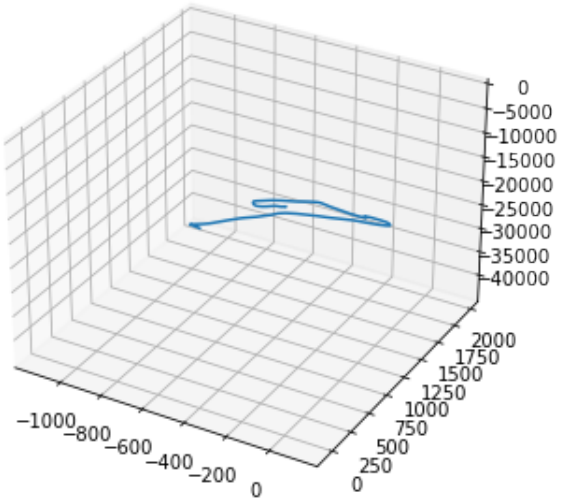


Figure 3.7: The path plotted by one of the algorithms. It is not completely accurate and must be improved.

Algorithm 3 Pedestrian dead reckoning

1: **procedure** PDR
2: **while** data **do**
3: $orientation \leftarrow get\_orientation\_angles(orientation, data, \alpha)$
4: $a \leftarrow linear\_accerleration(data)$
5: $\alpha \leftarrow angular_acceleration(data)$
6: $\alpha \leftarrow Modify\_frame\_of\_ref(\alpha, orientation)$
7: $a \leftarrow Modify\_frame\_of\_ref(a, orientation)$
8: $v \leftarrow v + a * t$
9: $p \leftarrow p + v * t + 0.5 * a * t^2$

Using the IMU the path of the person is roughly identifiable even though it is not perfectly accurate. But this will not be a problem as we do not require a lot of accuracy and if necessary the accuracy can also be improved through other hardware input like the LIDAR.

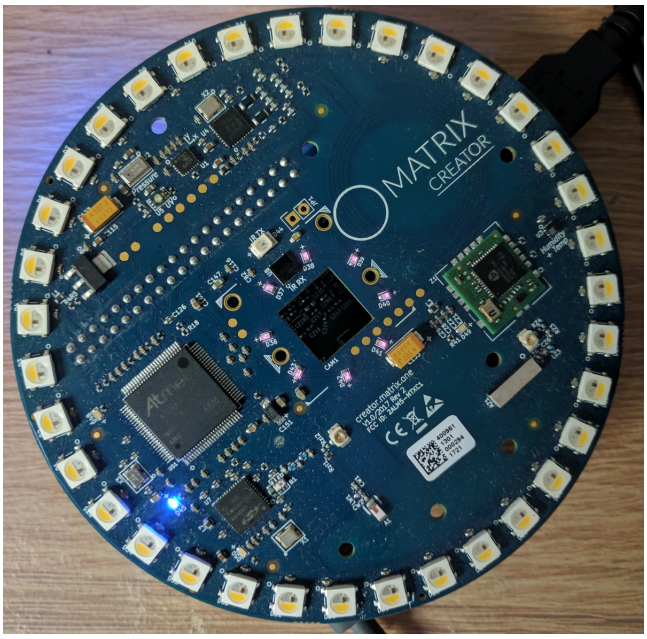


Figure 3.8: Microphone array

3.2.3 Microphone Array

The microphone array is a device that helps detect the Angle of Arrival (AoA) of the sound rays. Figure 3.8 shows an image of the same. It consists of microphones periodically placed, in our case circularly.Each of the microphone detects the sound and based on the phase difference the angle at which the sound is incident can be figured out. This helps to aid our graph search by allowing for the rays to be retraced using the floor plan from the point cloud information. The retraced rays allow for showing the path towards the source as intensity might not always be the best guiding modality due to reflections.

Figure 3.9: Raspberry Pi

3.2.4 Raspberry Pi

The device that ties all the above components together is the Raspberry Pi. It is a small board computer which can be connected to various components to create a device as per our requirement. A picture of this is shown by figure 3.9. It works as the CPU and by connecting it to a monitor, keyboard and mouse the code for the algorithm can be incorporated into the system. After this the various components are connected and using the IP address of the Raspberry Pi we SSH or VNC into it from another system. This allows us to disconnect the Raspberry Pi from the unnecessary devices, connect them to the components that we require which in our case are the LiDAR, IMU and Microphone array along with a mobile power source and control it through the other system. Using this we can make the various components of the system work together. In our case we have not developed the algorithm sufficiently to work independently by taking inputs from the components however we used each of the components separately connected to the raspberry pi to get data of a certain trajectory or positions and then use this data to first generate static algorithms of path before incorporating them into the overall code of the algorithm to take information in real time and act accordingly. The Raspberry Pi was booted using a version of the Raspbian operating system for this purpose.

CHAPTER 4

Solution Approach

4.1 Graph creation algorithm

With the information for the point at which the receiver is present, we used the point cloud to generate a graph Kisner and Thomas [2018] that gives the points to which the receiver can move. This involved two parts. The first step is to convert all the angles that have no point into contiguous gaps. The second step is to convert these gaps into nodes by finding the point to be traveled.

For the first step, we collected all the corresponding angles at which the contour does not have a point and then clustered them using the DBSCAN algorithm library. We used this method over others like `K-means` as in this application we know that the angles are not uniformly distributed but instead form a pattern of some number of clusters which are the openings. Thus, there is clear density difference, and we also do not know the number of clusters to make into as this could vary from position to position; DBSCAN is the most appropriate in this situation. However, this is not entirely robust and faces issues in particular circumstances. In cases where the majority of the surrounding space is empty, this algorithm fails to capture the essence of the situation. Even though it is all made into a single cluster, there may be different situations in different areas in that node. But this algorithm will only explore one of the directions in the mid, which will not do adequate justice to the situation. Thus, to improve this we assumed a threshold for the size of each node and if the cluster size exceeds this then it is subdivided

into smaller nodes each of the maximum allowed angular size. This allows for more effective search of the space but it is a tradeoff in the sense that it will take much longer for the search to complete. It is also difficult to calibrate the node size in different scenarios. In our algorithm we have calibrated it by putting an upper limit on the radial distance of the node which is taken to be the range of line of sight of the receiver. But this as we know need not be consistent in all situations.In some places the surroundings are clearer leading to better line of sight in which case larger sized nodes would suffice leading to waste of time, or on the other hand if the environment is smoky, etc, due to low line of sight smaller sized nodes would be more optimal. An alternative could be to use ways like hierarchical clustering among the angles in the node and choose the direction appropriately or split it further if necessary.

The clustered points are then used to create each node. To create a node, we need edge distance and child point. For the edge distance, we used the maximum threshold of the LIDAR but it is not the most accurate depiction. We plan to improve this by taking an average of the distances of the points just before and just after the gap and using that as the approximate distance towards this opening. This part is in progress. For the child point, we picked the mid angle among the cluster and found the point at a distance equal to the edge distance calculated earlier in that angular direction. We also stored the cluster of angles as part of the node to calculate the cost of each node for selective search. The children of each node are stored as an adjacency map. The pseudo code for this algorithm which updates the adjacency map at each stage to include the nodes that have been newly discovered, is shown by Algorithm 4. Thus, using this idea, the graph is created at each point visited by the receiver. In Figure 4.1, the progression of the graph generation after discovering new nodes at each of the points visited starting with

the initial receiver position in Fig 3.1 is shown.

Algorithm 4 Graph Generation

```
1:  procedure GET_GRAPH(point, gap_angles, contour)
2:      clusters ← DBSCAN(gap_angles)
3:      while cluster in clusters do
4:          max_angle ← max(cluster)
5:          min_angle ← min(cluster)
6:          mid_angle ← mid(cluster)
7:          edge ← (dist(point, contour[max_angle+1])+dist(point, contour[min_angle−
    1]))/2
8:          child_point ← point_at(mid_angle, edge)
9:          nodes ← nodes + [(cluster, edge, child_point)]
10:     adjacency_map[point] ← nodes
```

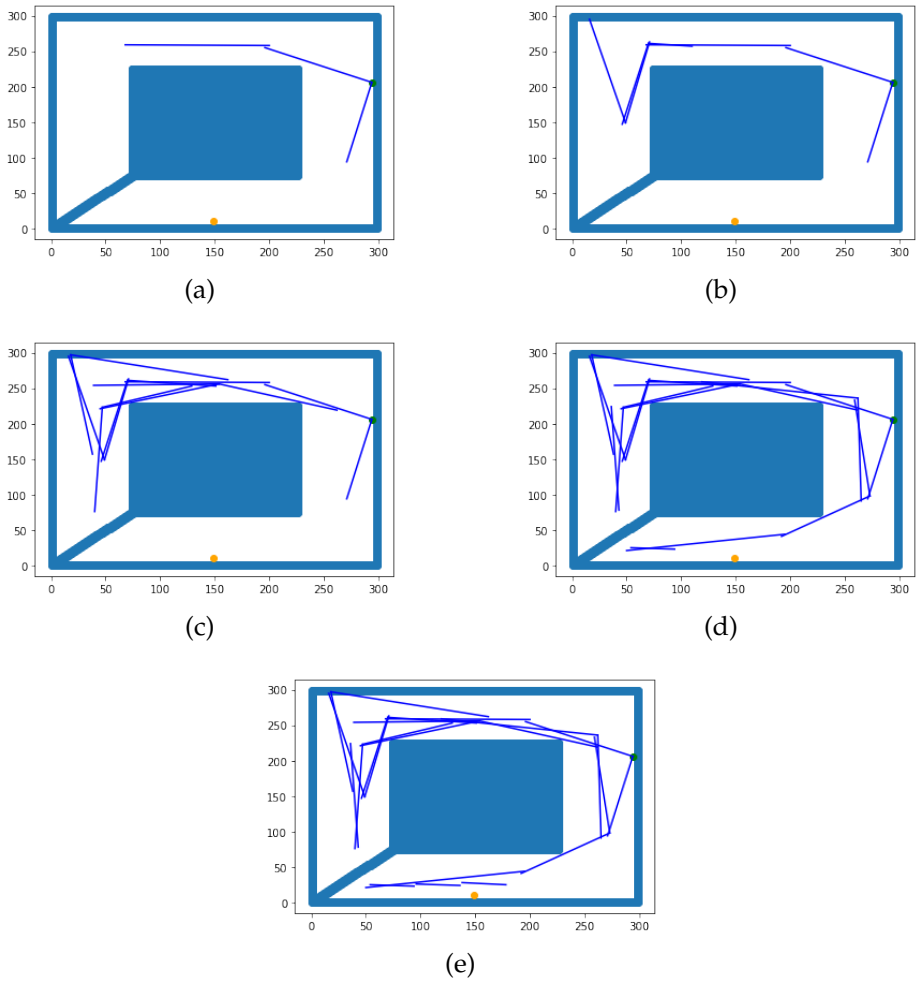


Figure 4.1: Indication of how the graph evolves at various point

The minor issue faced here was that when clustering angles, the angles 0 and

360 are next to each other, but this will not reflect properly in the clustering. These two will be made into separate clusters and need to be joined. Thus, to take care of this, we checked for clusters containing these values and if there was a node of both types, we removed those two and added a different combined node. We also appropriately took care of the start and angles as these would not just be the maximum and minimum in this case. In some situations where the receiver is too close to multiple walls or near corners the algorithm fails to recognise narrow paths sometimes and leads to resulting in no path being present. This calibration of maximum edge distances must be done carefully so that neither is time wasted in making nodes at short distances nor are narrow paths missed due to longer ranges.

4.2 Blind routing

Now that the graph has been created, the next step would be to search through the nodes and find a path towards the source. The amount of time taken to search using a method gives us an idea of its effectiveness. This includes the amount of distance traveled as well as the number of nodes visited. The first search that we seek to perform is a blind search. The receiver travels in any particular direction going forward to a new location until it either reaches the source or hits a dead-end, in which case it retraces its path until there is an unexplored path. This method is close to the implementation of a Depth First Search (DFS), and so we have used that method to observe a blind search. A DFS could be effective, but it depends on the direction in which the search starts and the positions of the source and receiver. To ensure that the child does not go back to the parent's direction, we restricted our visit to only those nodes that are not within a certain radius of the nodes that

have already been visited. This algorithm is shown in the form of pseudo code by Algorithm 5. Initially it was implemented iteratively however to enable ease of measuring backtracked path it was changed to a recursive implementation. There is a possibility that the receiver may not move to the exact child location wherein either there is some error or receiver may also go to a wildly different position. This is also taken into account by adding some error to the child location generated by the algorithm and assuming that the receiver moved to this place. For the setup shown in Figure 3.1, the path routed by this algorithm from the receiver to source locations is shown by Figure 4.2.

Algorithm 5 Point Cloud DFS

```
1:  procedure DFS(point)
2:      visited ← visited + [point]
3:      if reachable(point, source) then
4:          return
5:      while child of point do
6:          if any_reachable(child, visited) then
7:              continue
8:          contour ← point_cloud(child)
9:          angles ← gap_angles(contour)
10:         get_graph(child, angles, contour)
11:         DFS(child)
```

4.3 Selective routing

Now that the basic blind searching approach has been established, we could try to do something that performs better. A basic approach to improve this algorithm is to introduce a heuristic function that will help identify better child nodes, although this may not be completely reliable. For this, we took the intensity of sound on average in the area swept by the angular space and preferred the child with the most intensity. This may be skewed because of the reflections of sound along the

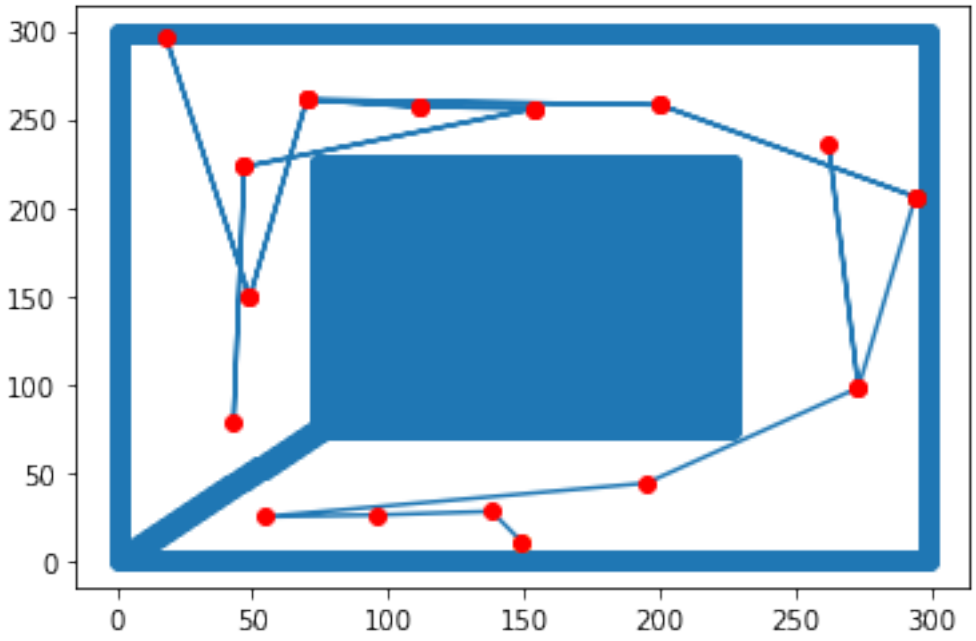


Figure 4.2: Path traced by Blind search. Time taken:603.2126338056737

path traveled and so depends on the obstacles. Nonetheless, it gives a measure of the direction from which the source is likely to be situated. To get a sense of the intensity in that direction, we calculated the average intensity (which we have from the system setup) of all the points in each of the angles in that gap cluster ranging from a minimum radius to a maximum radius, both of which are specified. The node which has the highest such value will be explored first. Here too we take into account that the receiver may not exactly move to the prescribed child node and for this some error is added to signify the new location. This modified version of the DFS algorithm is given by Algorithm 6 where the children are sorted by their heuristic values before being explored. For the setup shown in Figure 3.1, the path routed by this algorithm from the receiver to source locations is shown by Figure 4.3. We can see from the time taken as well as the path itself that selective search performs better than its counter part.

As the path finding depends on the number of nodes, it is essential to calibrate

Algorithm 6 Selective Routing Algorithm

```
1:  procedure SS(point)
2:      visited ← visited + [point]
3:      if reachable(point, source) then
4:          return
5:      children ← []
6:      while child of point do
7:          heuristic ← avg_intensity(child, intensity)
8:          children ← children + [(child, heuristic)]
9:      sort(children)
10:     while child of children do
11:         if any_reachable(child, visited) then
12:             continue
13:         contour ← point_cloud(child)
14:         angles ← gap_angles(contour)
15:         get_graph(child, angles, contour)
16:         SS(child)
```

the parameters properly to make sure that we do not over search or under search. This involves checking the values precisely. If the parameters are skewed the receiver may take too long to search for short distances or on the other hand search too little and inevitably miss the source location even though there is a path leading directly to the source, thus giving a result that there is no path. Both these situations are not ideal and must be avoided.

Although Intensity is a good guiding metric for the search, it is observed that sometimes especially when the receiver comes close to the source due to reflections and the area closest to the source being much more uniform, the search is not optimal and so to aid this we could use the AoA information.

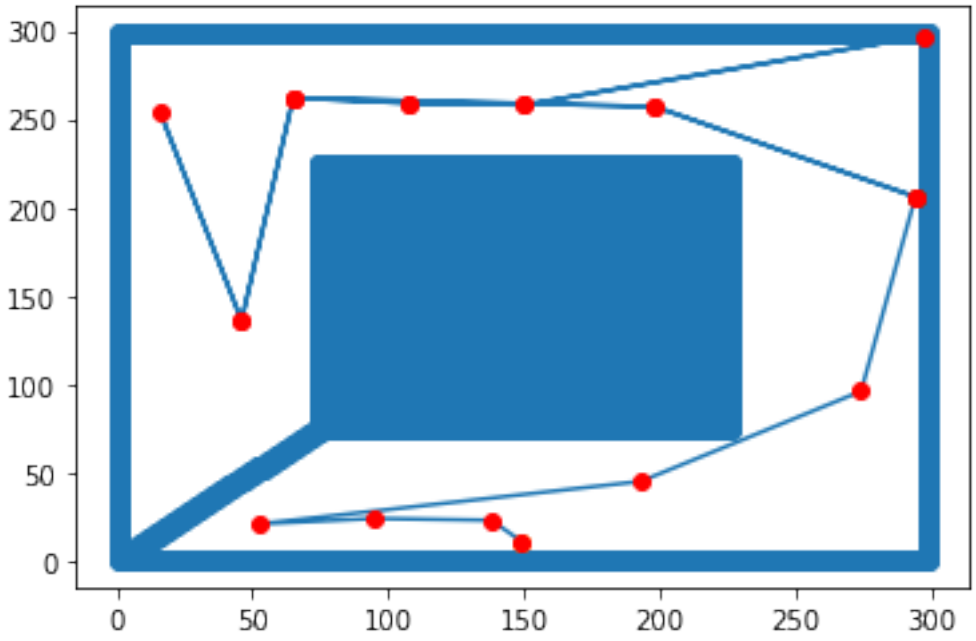


Figure 4.3: Path traced by Selective search. Time taken:481.4643850062261

CHAPTER 5

Evaluation

As mentioned in the motivation, the idea of this project is for applications where a person or robot may need to search for a person in a real-life scenario. Thus to effectively evaluate the setup and the algorithm, we need to consider cases inspired by real-life situations to observe the effectiveness. So we made some floor plans based on plans of actual buildings to observe how the algorithm works in each case.

The receiver and source were randomly allocated. To evaluate the effectiveness of the solution for each floor plan, we have generated the floor plan matrix along with source and receiver, a heat map that shows the intensity of sound at every point of the matrix that has been obtained as a result of the ray tracing algorithm, the graph generated by the corresponding algorithm, and the path generated by the DFS blind search algorithm as well as the selective search algorithm from source to destination. We have also created a function to measure the time taken by the person to reach the node, assuming he travels at a fixed speed and takes a fixed amount of time to scope the surroundings and decide on the new direction at each node (Both of these parameters are set by us). In the following cases, we have assumed the receiver to be traveling at a speed of 5 units, and at each node, it requires 5 units to gauge the direction it needs to travel to. The parameters set for these cases are such that maximum edge distance of the graph is 10 and this is also taken to be the range of line of sight of the receiver. The maximum distance a light ray travels is 10000 before it attenuates.

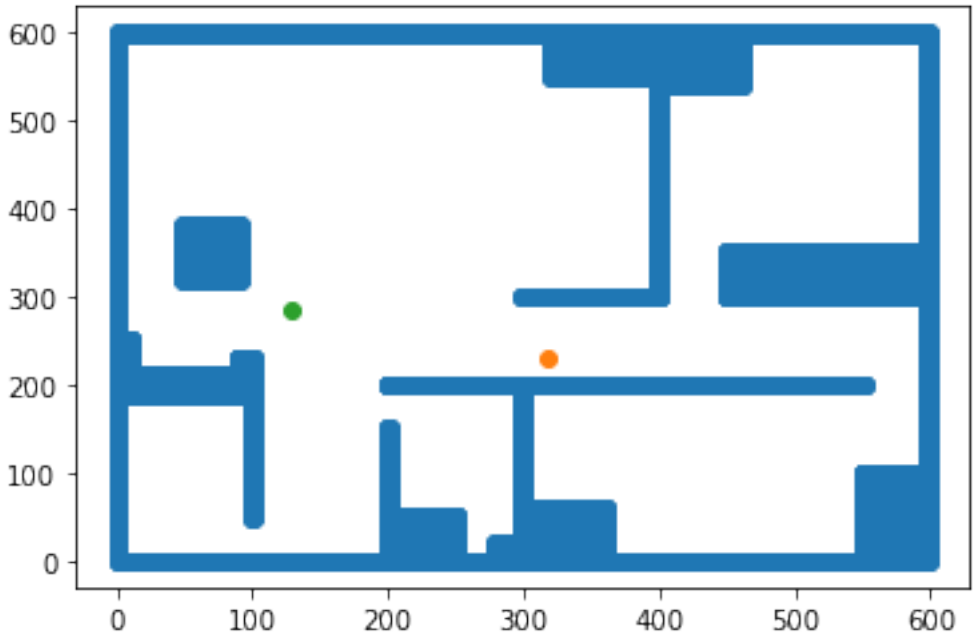


Figure 5.1: Floor plan of house. Source: (318, 229) Receiver: (128, 286)

5.1 House based case

5.1.1 Description

The first-floor plan is based on a house on a single floor. It consists of a living room cum dining area, two bedrooms, a kitchen, a laundry, and a bathroom. Furniture is depicted by rectangular obstacles of proportional size in the appropriate places. The size of the house is represented as a 600 x 600 matrix. The floor plan and initial positions of source and receiver are shown by Figure 5.1. The heat map of intensity of sound generated by the source at all points is shown by Figure 5.2. The graph and path generated by the receiver throughout the entire procedure is shown by Figures 5.3 and 5.4, and for selective search by figures 5.5 and 5.6 respectively. The label for the path figures indicates time taken for the entire search in that particular case.

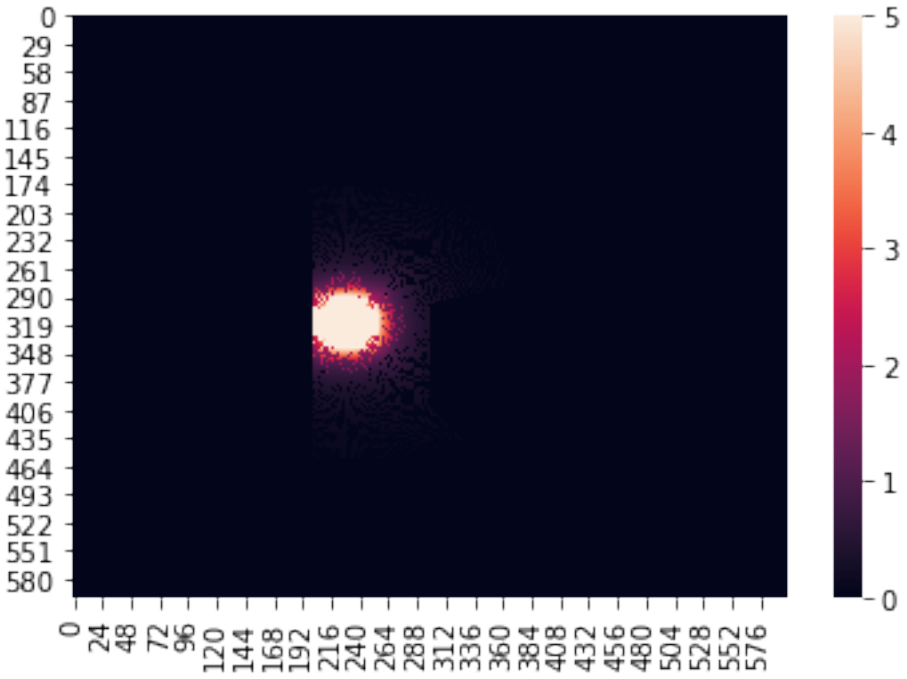


Figure 5.2: Heat map generated by source for the house plan(coordinate axes direction is slightly different)

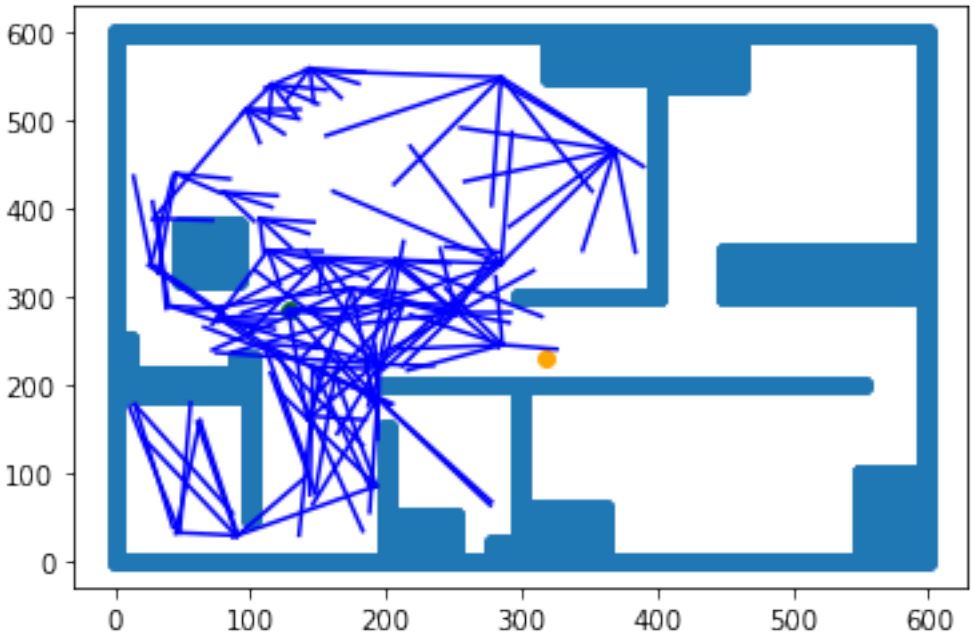


Figure 5.3: Graph generated along the Blind search path to find the source by receiver for the house plan

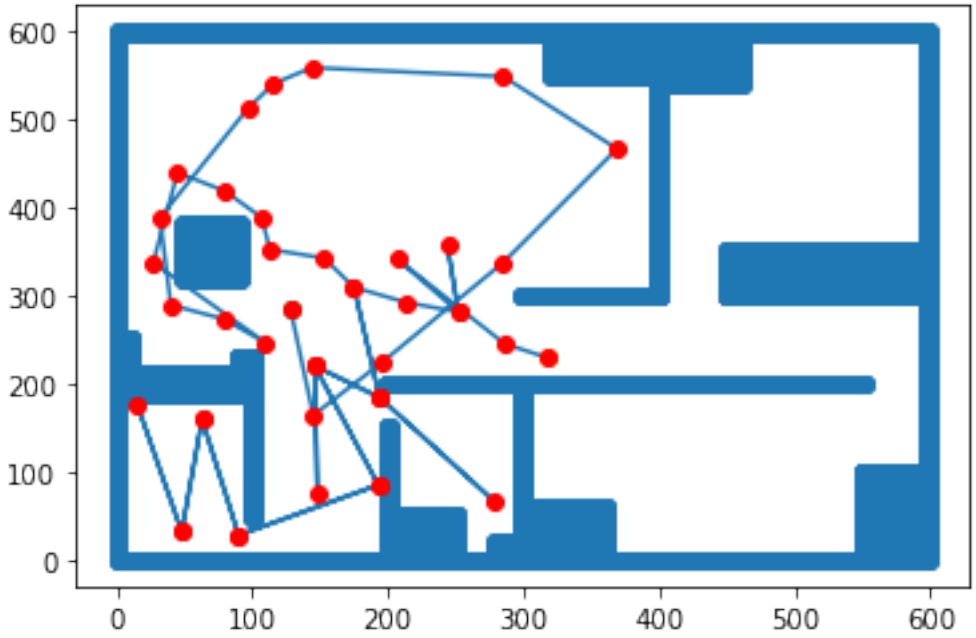


Figure 5.4: Path generated by Blind search for house
Time taken: 1086.110983851719

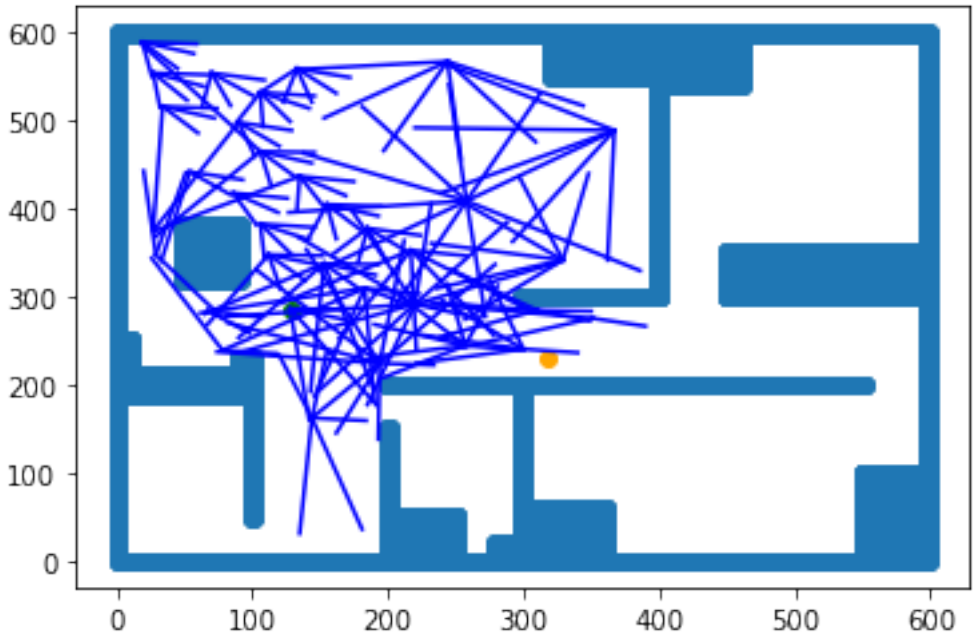


Figure 5.5: Graph generated along the Selective search path to find the source by receiver for the house plan

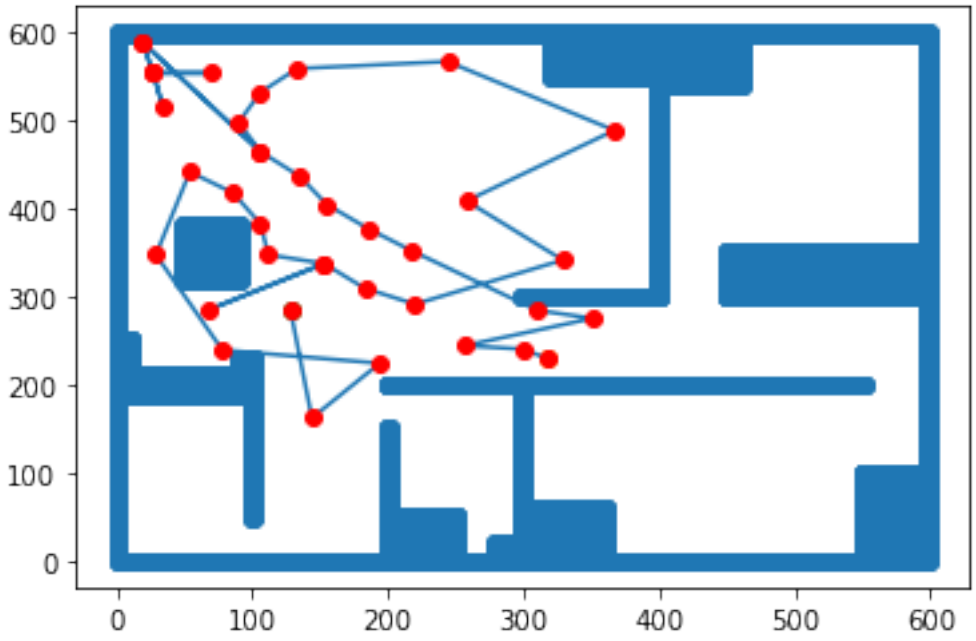


Figure 5.6: Path generated by Selective search for house
Time taken: 733.8738092843193

5.1.2 Observations

In this case, while the selective search performs better than the blind search, it is clear that neither is close to optimal. The difference in time taken is helpful but we can see that there is room for the algorithm to improve.

5.2 Restaurant based case

5.2.1 Description

The second plan that we have generated is based on the floor plan of a restaurant. It consists of an ample seating area, a restroom each for men and women, a kitchen, and an office room. In the seating area, rows of chairs and tables are depicted as a single rectangular obstacle, and it is assumed that it is not possible to move

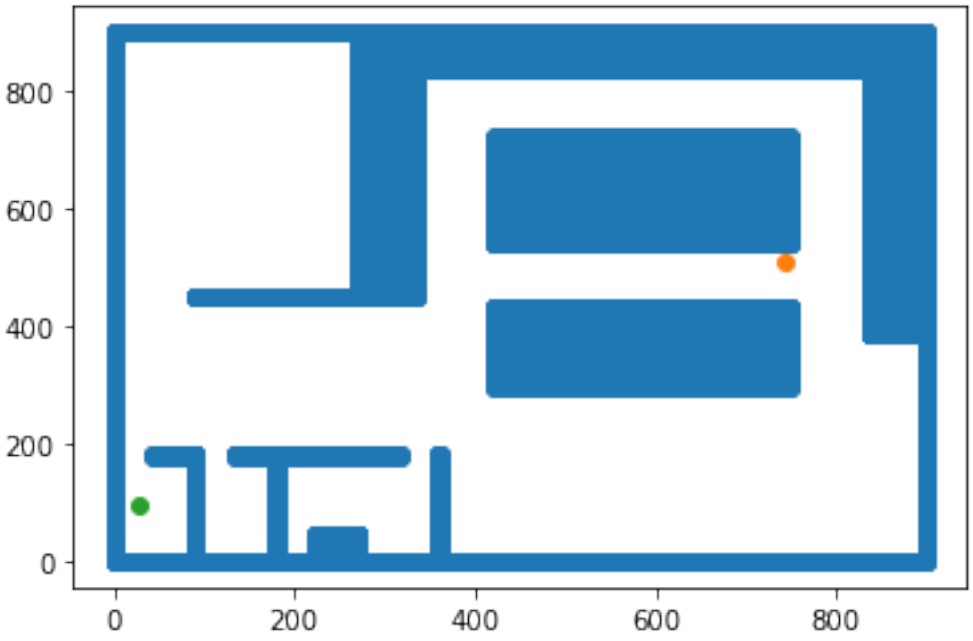


Figure 5.7: Floor plan of restaurant. Source: (743, 507), Receiver: (29, 95)

between tables or chairs within a row easily. The size of this restaurant is it as a 900 x 900 matrix. The floor plan of the restaurant and initial positions of source and receiver are shown by Figure 5.7. The heat map showing intensity of sound generated by the source at all points is shown by Figure 5.8. The final graph and path generated by the receiver throughout the entire DFS search is shown by Figures 5.9 and 5.10 respectively. The same for selective search are figures 5.11 and 5.12 respectively. The label of path figures indicate time taken for the search process.

5.2.2 Observations

Here, we see one of the cases where there is a tradeoff in the algorithm between effectiveness and correctness. Although the space is large, some of the gaps are very narrow. Due to this The maximum distance of each node must be adjusted to make sure that these small openings are not missed leading to the conclusion that

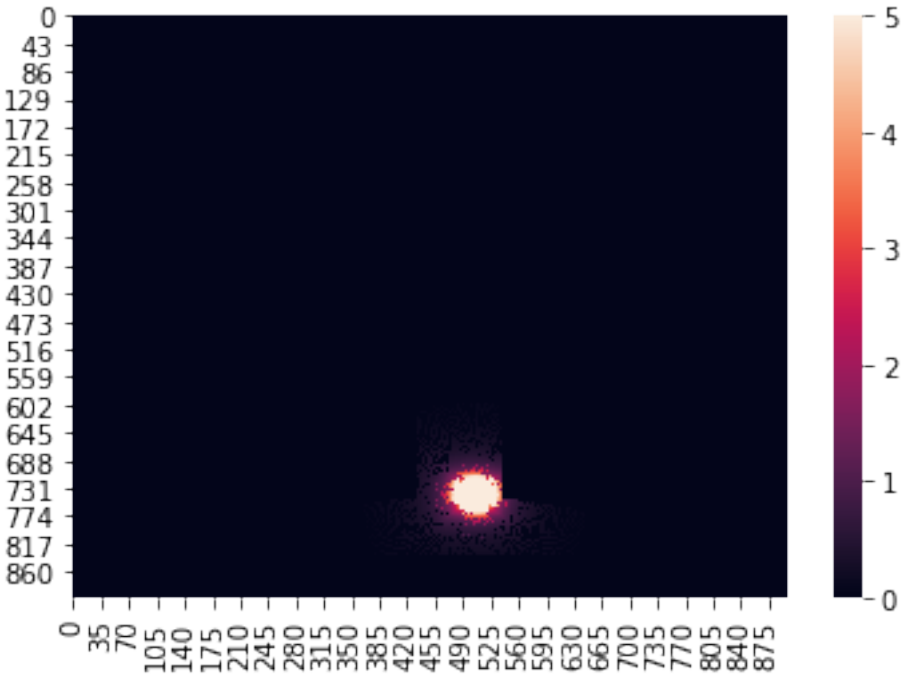


Figure 5.8: Heat map generated by source for the restaurant plan(coordinate axes direction is slightly different)

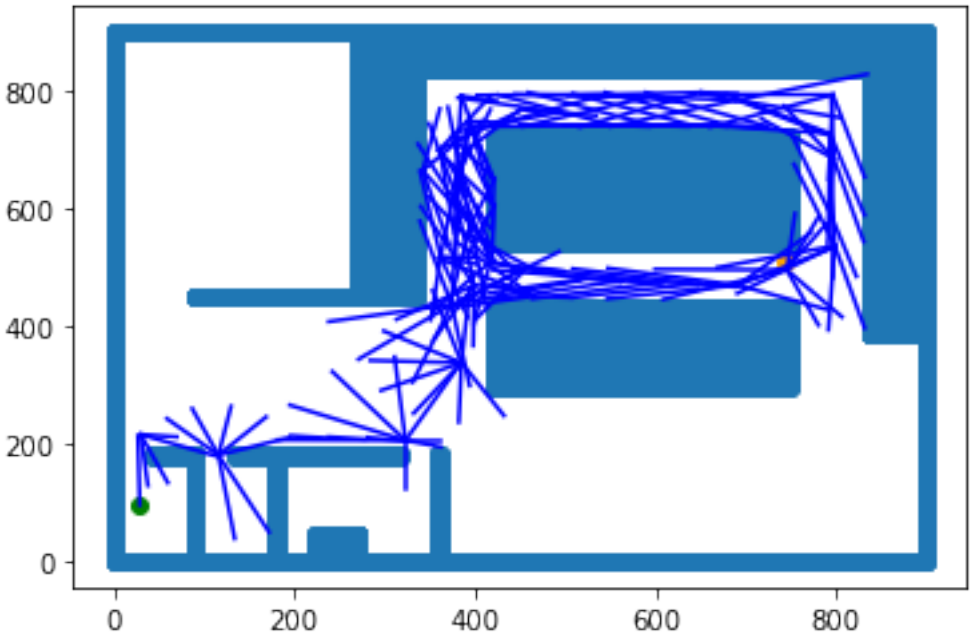


Figure 5.9: Graph generated along the Blind search path to find the source by receiver for the restaurant plan

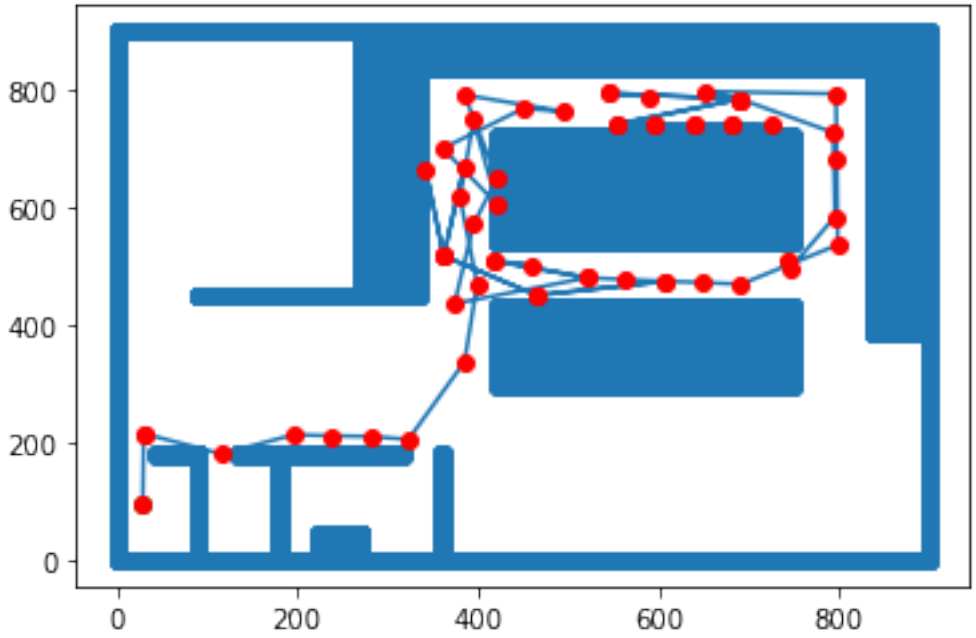


Figure 5.10: Path generated by Blind search for restaurant
Time taken: 1417.2850917865676

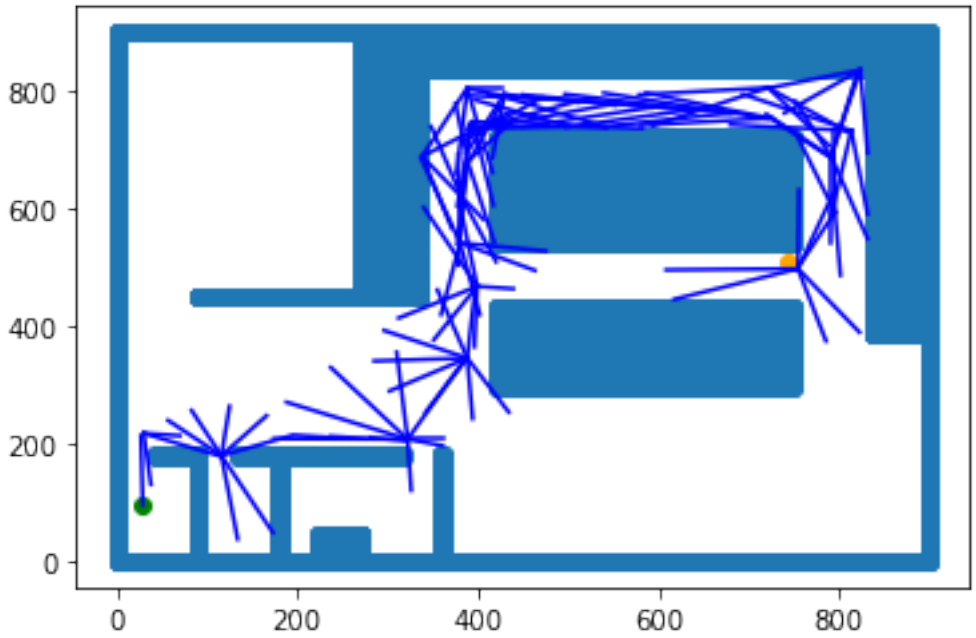


Figure 5.11: Graph generated along the Selective search path to find the source by receiver for the restaurant plan

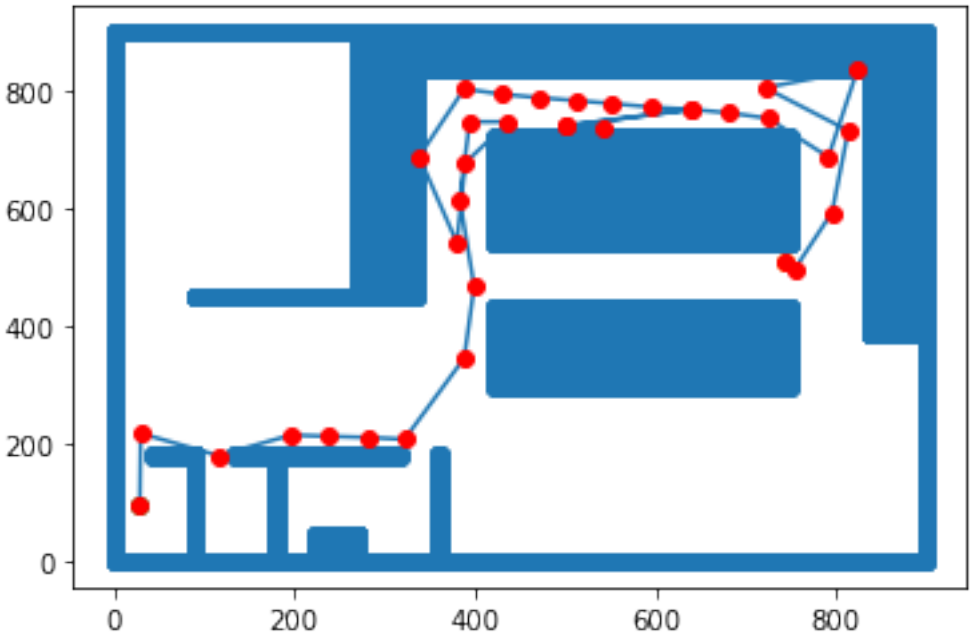


Figure 5.12: Path generated by Selective search for restaurant
Time taken: 763.0206947265667

there is no path even when ther exists one. However as we decrease the maximum edge size of the graph we waste more time to create children nodes in open spaces where larger distances can be covered without losing on correctness. The solution can perhaps be improved to modify the maximum edge distance based on its environment in which denser environments have shorter maximum edge distance and instead of directly taking an average of the start and end angle points which is not effective in narrow, long, corridor-like scenarios we could make heuristic based on the farthest part of the point cloud as well. The nodes divided based on threshold can have different edge distances instead of the same making it more effective as well.

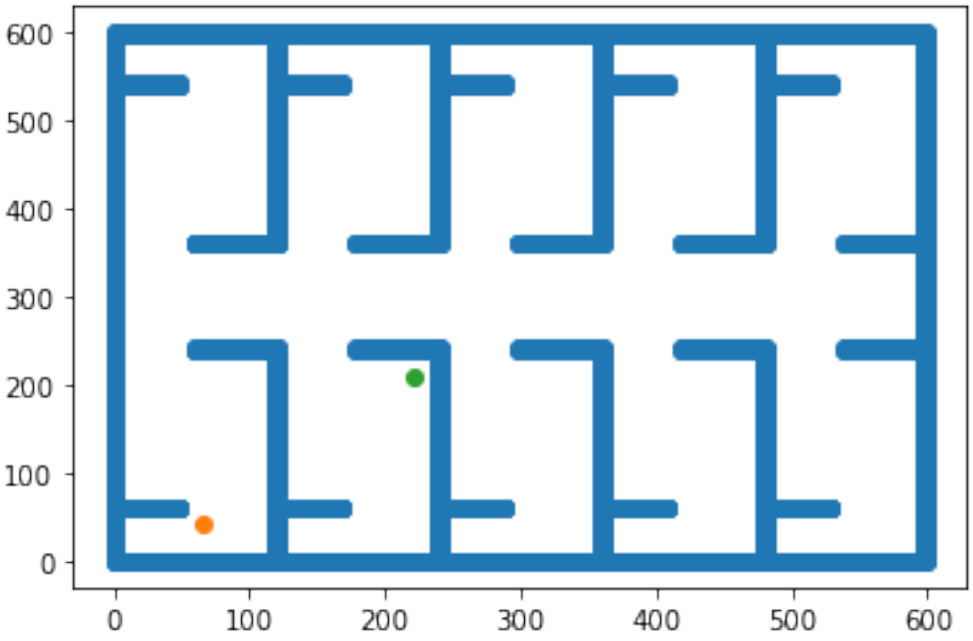


Figure 5.13: Floor plan of hotel. Source: (65, 43), Receiver: (222, 208)

5.3 Hotel based case

5.3.1 Description

The third and last plan is based on the floor plan of a hotel. It consists of an array of rooms on both sides of a narrow corridor. There is no connection between the rooms. There are ten rooms in total, five on each side of the corridor. Each of the rooms has a bathroom attached to it. The size of the whole floor is depicted in a matrix of size 600 x 600. The floor plan and initial positions of source and receiver for this case are depicted by Figure 5.13. Here, the heat map of intensity of sound generated by the source at all points is shown by Figure 5.14. The final graph generated by the receiver while searching for the source using DFS is shown by Figure 5.15 and the path traversed by the receiver during this search is shown by Figure 5.16 in which the label gives the time taken. The corresponding figures for selective search are given by figures 5.17 and 5.18 respectively.

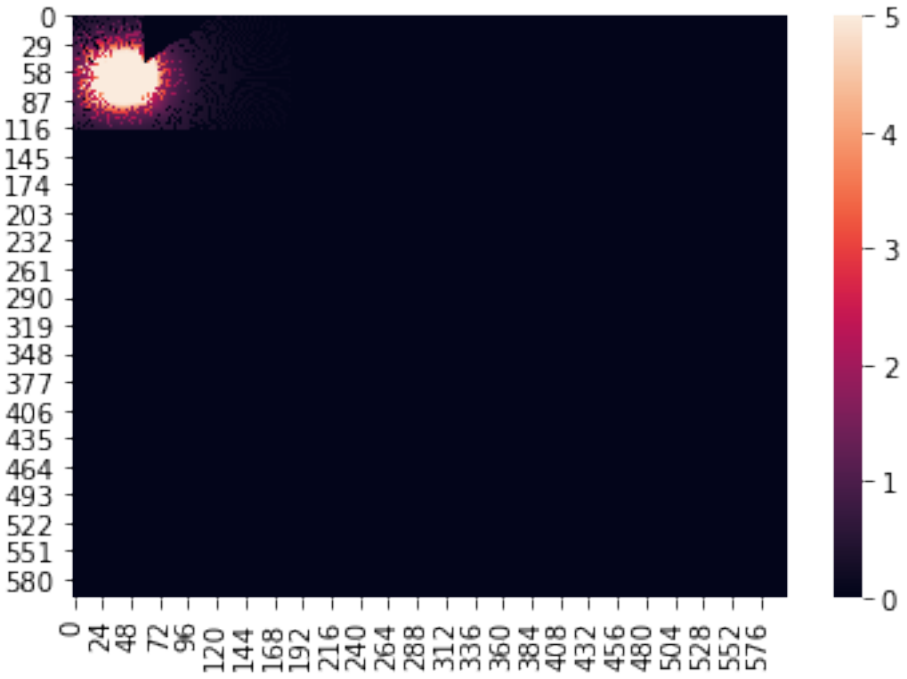


Figure 5.14: Heat map generated by source for the hotel plan(coordinate axes direction is slightly different)

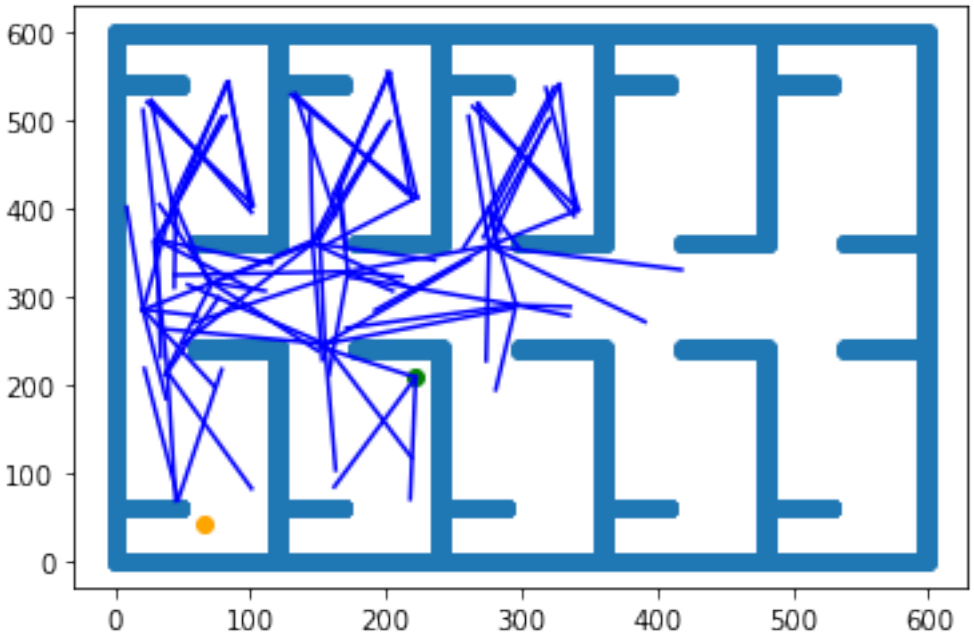


Figure 5.15: Graph generated along the Blind search path to find the source by receiver for the hotel plan

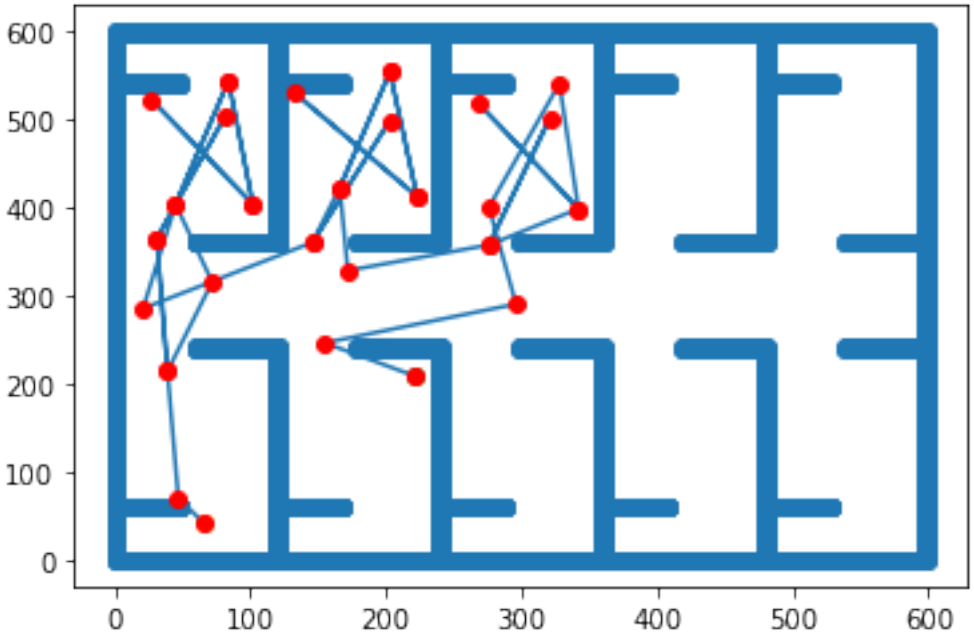


Figure 5.16: Path generated by Blind search for hotel
Time taken: 1145.4131543259061

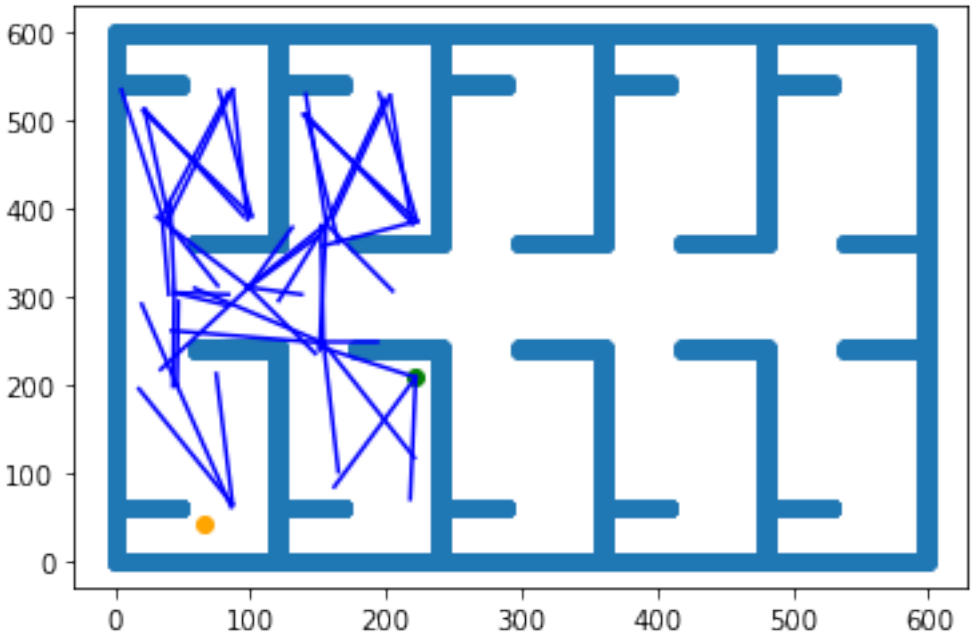


Figure 5.17: Graph generated along the Selective search path to find the source by receiver for the hotel plan

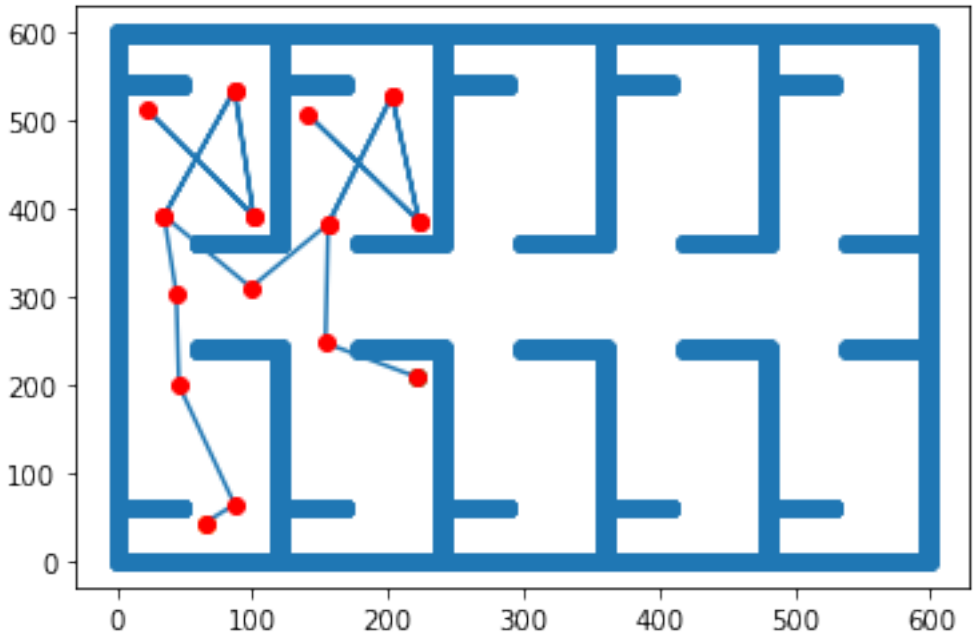


Figure 5.18: Path generated by Selective search for hotel
Time taken: 607.4297928670209

5.3.2 Observations

Here, we see that the selective search performs better but there seems to be some issue regarding the initial direction it is led to. There is a small issue in the light ray reflection simulation leading to slightly faulty intensities at the ends, which is being fixed and in some cases this leads the search awry. In situations like this where the place to be searched is quite large and intricate, and the signal is not so strong, it is difficult to map out a path depending on the signal alone. Thus, more efficient methods to quickly search an area are required to minimize the total time taken. In our case, it is more important to reach the source quicker than to reach it through the shortest path. Even if the shortest path is employed to reach the source, it may have taken the receiver longer to search other paths that did not lead to the source.

CHAPTER 6

Future steps for the solution

Since the selective search method is also not entirely robust, especially in situations with more obstacles or when the source signals are uniform in all directions, there could be a better improvement to this solution. We want to use the angle of arrival and point cloud to retrace the path in some way or to use the collective AoA of multiple rays to indicate openings so that the source could be localized. This would give a more accurate measure of the direction towards the source.

Another improvement, as mentioned earlier, is to use some measure so that the direction traveled within each node(cluster) is also decided instead of traveling blindly in the mid.

The selective search seen above is especially useless in cases where the sound signal does not adequately reach the receiver, and so it performs the same as DFS in that case. Thus, we need some ways to explore the nodes more effectively until it reaches the point where it can be guided by the signal. Therefore, another heuristic that could help is to use the actual edge distances and explore nodes that are nearer first, so that many of the closer nodes can have been visited in a shorter time.

Another reason for the graph structure to be improved is because, as seen in the DFS and selective search graphs, although it follows the optimum path in the graph, it is still not along a straight line, but rather, it wavers back and forth, leading to additional and unnecessary coverage of distance.

There must also be some dynamic nature introduced in the maximum edge

distance parameter selection as well as line of sight parameters as these depend entirely on the environment and this can be gauged using the data that we sensed along the course of the usage.

Another improvement to the algorithm is to use a reward based search system rather than the greedy approach based on a heuristic. In this case it is clear that neither AoA nor intensity information is accurate and so using greedy selective search may not be the best scenario. Thus using a reward based situation this can be taken care of in a more optimal fashion.

Apart from these steps, the solution can also be parallelized so that aspects like the ray tracing could be done concurrently and thus result in quicker answers.

The hardware aspect of the algorithm must also be finished so that the effectiveness of the solution can be observed in practice rather than on simulated data.

CHAPTER 7

Conclusion

Thus far, among the steps mentioned earlier of creating the graph and then searching through that efficiently, we have explored the first part and formulated an idea that captures the essence of what is required. Although more improvements can be made, it is a foundation for improvements and further developments. The algorithm for the same has been written for the simulated situation. The basic ideas of that second part have been implemented and evaluated for the simulation but there are more improvements that can be done using more modalities to guide the search that will significantly improve the performance. At present, the graph search while being effective is not robust at all times and with the aid of more information to have a better guiding heuristic it can be enhanced. Better evaluation metrics randomising the source and receiver location to explore the effectiveness of the algorithm in a more general sense where also implemented however there are some issue in this and so it was not included in the report. The practical hardware for this part has also been procured and of the various components we worked with the IMU, LiDAR and Raspberry Pi to generate data for various paths and then we worked on this data to research ways in modifying data to get the required information that suits the needs of our problem.The basic research and ideation of the necessary algorithms for the various components was achieved but the implementation is in progress. Work has also been done to modify the simulation algorithm for the incorporation of actual real time data after filtering and fine tuning for our benefit but this aspect is also in progress.

REFERENCES

Barabell, A., Improving the resolution performance of eigenstructure-based direction-finding algorithms. *In ICASSP'83. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 8. IEEE, 1983.

Dissanayake, M., **P. Newman**, **S. Clark**, **H. Durrant-Whyte**, and **M. Csorba** (2001). A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotics and Automation*, **17**(3), 229–241.

Kisner, H. and **U. Thomas**, Segmentation of 3d point clouds using a new spectral clustering algorithm without a-priori knowledge. *In VISIGRAPP (4: VISAPP)*. 2018.

Li, Q., **R. Li**, **K. Ji**, and **W. Dai**, Kalman filter and its application. *In 2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*. 2015.

Liu, Y., **K. Guo**, and **S. Li**, D-zupt based pedestrian dead-reckoning system using low cost imu. 2016.

Pradhan, S., **G. Baig**, **W. Mao**, **L. Qiu**, **G. Chen**, and **B. Yang** (2018). Smartphone-based acoustic indoor space mapping. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, **2**(2), 1–26.

Schmidt, R. (1986). Multiple emitter location and signal parameter estimation. *IEEE transactions on antennas and propagation*, **34**(3), 276–280.

Shen, S., **D. Chen**, **Y.-L. Wei**, **Z. Yang**, and **R. R. Choudhury**, Voice localization using nearby wall reflections. *In Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 2020.

Wang, M., **W. Sun**, and **L. Qiu**, {MAVL}: Multiresolution analysis of voice localization. *In 18th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 21)*. 2021.