IMPLEMENTATION OF A PATH FINDING ALGORITHM FOR THE NAVIGATION OF VISUALLY IMPAIRED PEOPLE

A MASTER'S THESIS

in

Computer Engineering

Atılım University

by

MUSTAFA KILINÇARSLAN

SEPTEMBER 2007

IMPLEMENTATION OF A PATH FINDING ALGORITHM FOR THE NAVIGATION OF VISUALLY IMPAIRED PEOPLE

A THESIS SUBMITTED TO
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE
OF
ATILIM UNIVERSITY
BY

MUSTAFA KILINÇARSLAN

IN PARTIAL FULLFILMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF COMPUTER ENGINEERING

SEPTEMBER 2007

Approval of the Graduate School of Natural and Applied Sciences

Prof.Dr. Selçuk Soyupak
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof.Dr. İbrahim Akman
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Asst.Prof.Dr. Çiğdem Turhan
Co-Supervisor

Asst.Prof.Dr. Nergiz E. Çağıltay
Supervisor

Examining Committee Members

Asst.Prof.Dr. Çiğdem Turhan _____

Asst.Prof.Dr. Nergiz E. Çağıltay _____

Asst.Prof.Dr. Ruşen Öktem _____

Asst.Prof.Dr. Murat Koyuncu _____

Asst.Prof.Dr. Mehmet Cantürk _____

ABSTRACT

IMPLEMENTATION OF A PATH FINDING ALGORITHM FOR THE NAVIGATION OF VISUALLY IMPAIRED PEOPLE

Kılınçarslan, Mustafa

M.S., Computer Engineering Department

Supervisor: Asst.Prof.Dr. Nergiz E. Çağıltay

Co-Supervisor: Asst.Prof.Dr. Çiğdem Turhan

September 2007, 62 pages

For visually impaired people it is always a problem to manage their life like a normal person. Even some very simple jobs, like going to job/school, shopping are sometimes causing big problems for them. There are several studies for visually impaired people to make their life easier which free them from boundaries. Unfortunately, these projects are very limited to integrate them to lead a normal life. In this study, a path finding algorithm for the navigation problem of visually impaired people is implemented. A shopping center is chosen as a case study for this research and the algorithm is tested on the developed laboratory environment. The tests are based on simulations performed by the data collected by means of the laboratory environment.

Keywords: Path Finding Algorithm, A* Algorithm, Visually Impaired Navigation

ÖZ

GÖRME ENGELLİLERİN YÖNLERİNİ BULMASINDA KULLANILMAK ÜZERE BİR YÖN BULMA ALGORİTMA UYGULAMASI

Kılınçarslan, Mustafa

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Yrd.Doç.Dr. Nergiz E. Çağıltay

Ortak Tez Yöneticisi: Yrd.Doç.Dr. Çiğdem Turhan

Eylül 2007, 62 sayfa

Görme engelli bireyler için hayatlarını normal bir birey gibi idare etmek her zaman bir problem olmaktadır. İşe ya da okula gitmek veya alışveriş yapmak gibi engelli olmayan bireyler için çok basit olan işler, görme engelli bireyler için sorun oluşturabilmektedir. Görme engelli bireyleri bu sınırlardan kurtarıp yaşamlarını daha kolay hale getirmek için yapılmış çeşitli çalışmalar bulunmaktadır. Ancak yapılan bu çalışmalar bireylerin normal yaşama entegrasyonu aşamasında yeterli ve etkin çözümler oluşturamamaktadır. Bu tezde, görme engelli bireylerin yönlendirilmesi probleminde kullanılmak üzere, A* yön bulma algoritmasının uygulandığı bir yardımcı uygulama geliştirilmiştir. Geliştirilen sistem, bir alışveriş merkezi olarak hazırlanan bir laboratuvarda bir durum çalışması olarak test edilmiştir. Testler için laboratuvar ortamında toplanan verilerin kullanılması ile gerçekleştirilen benzetimler esas alınmıştır.

Anahtar Kelimeler: Yön Bulma Algoritması, A* Algoritması, Görme Engelli Yönlendirmesi

To my dear wife

Thanks for her endless support

ACKNOWLEDGEMENTS

First, I would like to thank my thesis supervisor Asst.Prof.Dr. Nergiz E. Çağıltay and my thesis co-supervisor Asst.Prof.Dr. Çiğdem Turhan for their guidance, insight and encouragement throughout the study.

I should also express my appreciation to examination committee members Asst.Prof.Dr. Ruşen Öktem, Asst.Prof.Dr. Murat Koyuncu and Asst.Prof.Dr. Mehmet Cantürk for their valuable suggestions and comments.

Finally I would like to express my thanks to my wife for her endless assistance, encouragement and all of my family members for their patience, sympathy and support during the study.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

RFID	: Radio-Frequency Identification
RSS	: Radio Signal Strength
CPS	: Central Processing System
PFA	: Path Finding Algorithm
DARPA	: Defense Advanced Research Projects Agency
GPS	: Global Positioning System
DGPS	: Differential Global Positioning System
HHNU	: Hand Held Navigation Unit
OS	: Operating System
OaO Part	: Option and Operation Part

CHAPTER 1

Introduction

For vision disabled people (or visually impaired persons), it is always a problem to manage their life like a normal person. Even some very simple jobs like going to job/school, shopping are sometimes causing big problems for them. Before managing their life, they have also problems with obtaining the necessities of a human body. They can overcome these difficulties by themselves or with the help of a guide. There are several studies for visually impaired people to make their life easier which free them from boundaries. Unfortunately, these projects are not enough or efficient to integrate visually impaired people to lead a normal life, and there is not any working real life applications.

In this study, we tried to make an implementation of a path finding algorithm for navigating visually impaired people. The process of shopping is chosen as a case study for this research. Our aim is to develop a general approach so that in later studies, it will be possible to implement this work for navigating a visually impaired person in a campus, a town or a city. Currently this thesis study is part of a project "PUSULAL: Görme Özürlüler İçin Alışveriş Merkezlerinde Yardım Pusulası" funded by The Scientific & Technological Research Council of Turkey (TÜBİTAK), Project Number: 105E130.

PUSULAL consists of a generic compass developed for visually impaired people; RFID (Radio Frequency Identification) technology, wireless communication systems, a central processing system (CPS) and a voice assisted guiding system. In PUSULAL, when a visually impaired person (user) comes to the market, a navigation unit will be provided to her/him. By listening to the menu items, user will

be able to select a target location. Then the navigation unit will send the user's location and destination information to the CPS. The scope of this thesis will cover only analyzing the RFID RSS (Radio Signal Strength) data coming from the navigation unit and providing information about the directions for the destination point after the result of the path finding process. The Hand Held Navigation Unit (HHNU) that will be used in the system are being developed by other project members. Navigation unit has a special compass system whose directions can be sensed by fingers, and following the direction information from the compass, user will be guided to the related section of the market. Data coming from the navigation unit will be the input for our software and implemented algorithm will calculate the path between points. Then the route data will be sent back to the navigation unit. This process will be done every time user sends a new destination location.

There will also be some predefined buttons on the navigation unit, in case of emergency or any other situations. Users may want to stop the shopping and go for the payment, or they may want to go to the restroom, medical advisor etc. In these cases our algorithm will calculate and send the exit/evacuation path to the user unit.

Our path finding system will do the following jobs:

- Collecting the location information and the selected destination address from the user,
- Calculating the shortest path between the starting and final points,
- Sending the path information to the navigation unit.

1.1 Scope of the Study

PUSULAL is a help system that has been developed for the use of visually impaired people in order to help them for shopping by themselves. It uses RFID technology to find the location of the user in the market and to direct the user to the target location.

This thesis study covers the administrator module of the PUSULAL. Accordingly, the scope is to build and implement a path finding algorithm to a predefined map,

then step by step improve the system and develop it to the final application area. Main problem in navigation is finding the way/path and the algorithm that is used while developing the system. As we will see in the next chapter various algorithms are used in the literature for different application areas in detail. Algorithms are commonly used in computer games, robot motion/navigation, driverless vehicles, transportation networks and human navigation. One of those algorithms namely A* is implemented in this study in order to guide the user in the shopping area. Additionally, a user interface is developed to monitor and simulate the Path Finding Algorithms to any indoor area.

The organization of this thesis is as follows. Chapter 2 presents the literature survey which includes the studies on the path finding algorithms and the visually impaired navigation. The reason why we selected the A* Algorithm is also explained in 2nd Chapter. Chapter 3 explains the research methodology we used. Chapter 4 includes the details and information about the usage of the developed utility and there are comparative data samples from the software. These samples are made by simulating the test area in the utility and by using, sometimes modifying the method, the real RFID measurements. Integration of our system to the other parts of the PUSULAL project is also in this chapter. Finally, in Chapter 5 we make an evaluation of our study and give some feedback for other people who will work on similar subjects.

CHAPTER 2

Path Finding Algorithms

Path finding in the basic definition is moving an object from its initial position to the final location. Path Finding Algorithms (PFA) are used in many different application areas such as:

- Games and Virtual Tours
- Robot Motion and Navigation
- Driverless Vehicles
- Transportation Networks
- Human Navigation

In this section we will summarize some of these applications.

2.1 Application Areas of Path Finding Algorithms

Path finding algorithms have many usage areas. These algorithms are useful in the field of robotics, since they can be used to guide a robot around difficult terrain without human intervention [1]. This would be useful if the robot were on another planet like Mars, where some terrain must be avoided, but due to the extreme distances involved, controlling it completely via remote control would be impossible (too much delay in the radio transmission). It could also be useful if the robot were to operate underwater, where radio waves could not get to it [2]. Path finding algorithms could also be used in almost any case where a vehicle needs to go somewhere, while avoiding obstacles, without human intervention. Another use is in computer games where something needs to be moved from one place to another, avoiding any walls or other obstacles in the way [3].

These algorithms could also be used to find the shortest path to drive between two points on a map, the best way to route an e-mail through a computer network, or the shortest path to run telephone wires through existing circuits.

2.1.1 PFAs in Games and Virtual Tours

Path finding is a very important part of game programming. Game characters move according to the path they (or computer) calculate. There are some algorithms used in games changing with the complexity and purpose of the path calculation. The most used algorithm in today's games is the A* Algorithm.

> "What makes the A* algorithm so appealing is that it is guaranteed to find the best path between any starting point and any ending point, assuming, of course, that a path exists." [3]

Users navigating in a virtual building or the virtual copy of a real building, like a museum, can see the artworks without going to physical place. This is good for avoiding traveling long distances and reaching more people to show the works. Users can plan their visit path and algorithm calculates the route, then the virtual tour begins. Latiff and Hassan [4] worked on this scenario and developed an application capable of doing path planning process. They have used A* algorithm to find the path. The final path can be seen on a 3D environment (Figure 2.1).

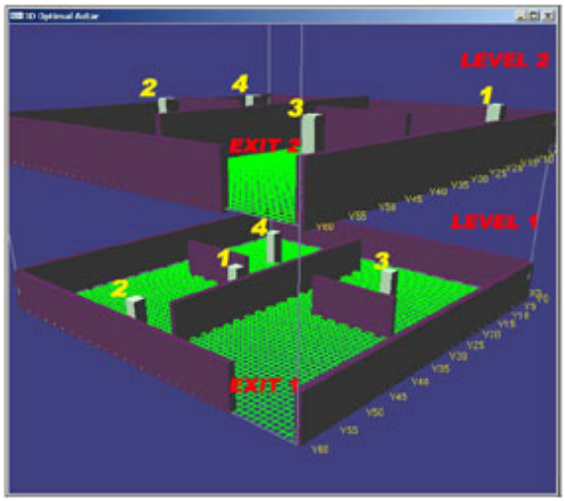


Figure 2.1 3D Model of Latiff and Hassan

Autonomous navigation of virtual characters is also an important task to work on. It is required to prevent from obstacles and continue to their routes without any damage. In a virtual environment, unlike the robotic navigation, there is no data coming from sensors, there exist the scene database only. Fröhlich and Kullmann [5] have used A* algorithm, studied on the environmental modeling of the area and concluded that the uniform-sized grid cell approach is a better choice. They also stated that A* algorithm works well on the environments where the world is flat, that suits our environment well.

2.1.2 Robot Motion and Navigation

Mobile robots moving in indoor or outdoor environments should have their route planning and navigation systems in order to be able to find their way. Generally the navigation and path finding units are placed on the robots, so that they can move by themselves. It is important to have this property, if we think about the usage of these robots. In military, detection of hazardous or explosive materials and discovery and investigation of unknown areas are very suitable for this type of robots.

Razavian and Sun [6] proposed a new algorithm, named Cognitive Based Adaptive Path Planning Algorithm (CBAPPA) and compared it with A* Algorithm and Focused D* Algorithm (D*, Dynamic A*, algorithm is derived from the A* Algorithm and has some improved capabilities used for autonomous units). They used it for observed behaviors of the biological units and paid attention to the

behavior of ignoring the irrelevant information from surroundings and trying to reach the target quickly. This process may not use optimum paths but the results were efficient. This algorithm is also prepared for self-processing units, which can be a good example for our future works.

Koenig and Likhachev introduces another new algorithm called "D* Lite" which is a variant of D* Algorithm for improved fast replanning in unknown terrain [7]. D* algorithm performs better than A* in unknown areas when used in autonomous robots. D* Lite is more efficient than D*, since it is simpler, shorter and easier to understand. D* and D* Lite algorithms update their knowledge about the terrain when they move in it, and continuously perform replanning of the path. This approach is quite different from our environment, but can be very useful when we start to implement our system in different (may be unknown) environments.

It is stated in Artificial Intelligence (AI) for Game Developers book [3] that: "Path finding problem is thought as solved, future works are for making the algorithm fast and more efficient." Studies give us more efficient and faster algorithms to use and implement in different areas.

2.1.3 Driverless Vehicles

Another application area of the path finding algorithms is the automated vehicles that will be able to find its way without making interaction with the obstacles. Once again its major usage is in military, but it can be implemented to different areas of our life. In the future, they can search for the lost outdoor adventurers, hikers, or they can be very useful in case of natural disasters, with the sensors they have.

Yahja, Singh and Stentz [8] also introduced us with the D* algorithm which stands for Dynamic A*. They used framed-quadtrees (Figure 2.2) when constructing their environment and data structures, and they tested their algorithm in an autonomous vehicle (Figure 2.3). Their algorithm continuously modifies itself when it gets new information about the terrain.

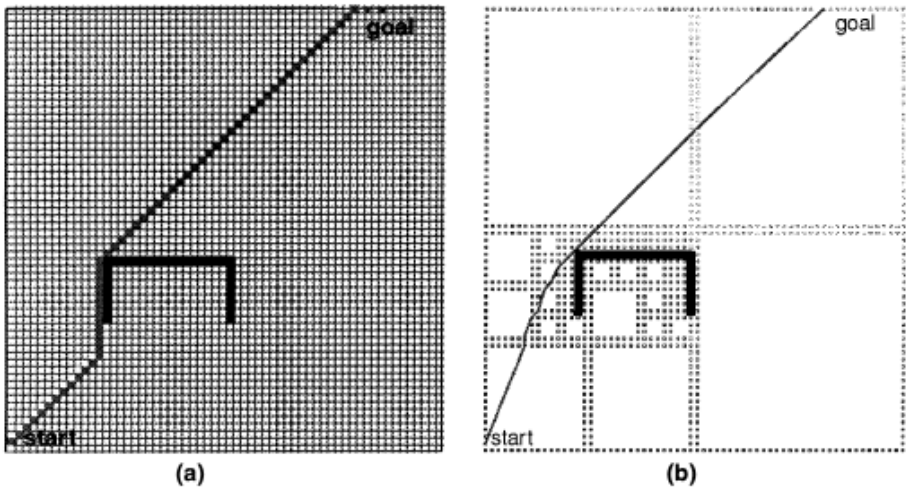


Figure 2.2: (a) Grid vs (b) Framed-Quadtree approach in D* Algorithm

They have pointed that :

> "If complete and accurate maps were available, it would be sufficient to use a standard search method such as A*." [8]

Figure 2.3: Autonomous vehicle used in D* algorithm test

In the United States there is a race organized by The Defense Advanced Research Projects Agency (DARPA), whose participants are the autonomous ground vehicles from all over the world. Race was held twice until 2007; one in 2004 in which none of the participants could be able to complete the course and the second one was in

2005. In 2005, 4 vehicles finished the 132-mile desert track and the winner is a VW Touareg (Figure 2.4) developed by Stanford Racing Team (http://www.stanfordracing.org). This car has a processing system to calculate its route while on the road. Onboard computers control the vehicle from start to the end, and there wasn't any intervention from the race team. It continuously modifies the path according to the information it gets from its sensors while it moves on the route,.

Figure 2.4: Winner of the DARPA Challenge 2005

2.1.4 Transportation Networks

In road networks it is getting more important to find the way to the destination point. If a person is new to a place, much time can be wasted until finding the destination. There exist some products developed to overcome this difficulty, providing a map of the region. After entering the starting point and the final location, it is possible to get the shortest path.

Experimental studies are performed to select the best algorithm for using in path finding process. For example, Jacob, Marathe and Nagel [9], from Los Alamos National Laboratory, have compared the shortest path algorithms by doing experimental analysis on a big database (Figure 2.5). They evaluated the Dijkstra, A* and modified A* algorithms with respect to the computation time on real networks, solution quality, ease of implementation and the extensibility of the algorithm. They found out that A* algorithm has better time efficiency.

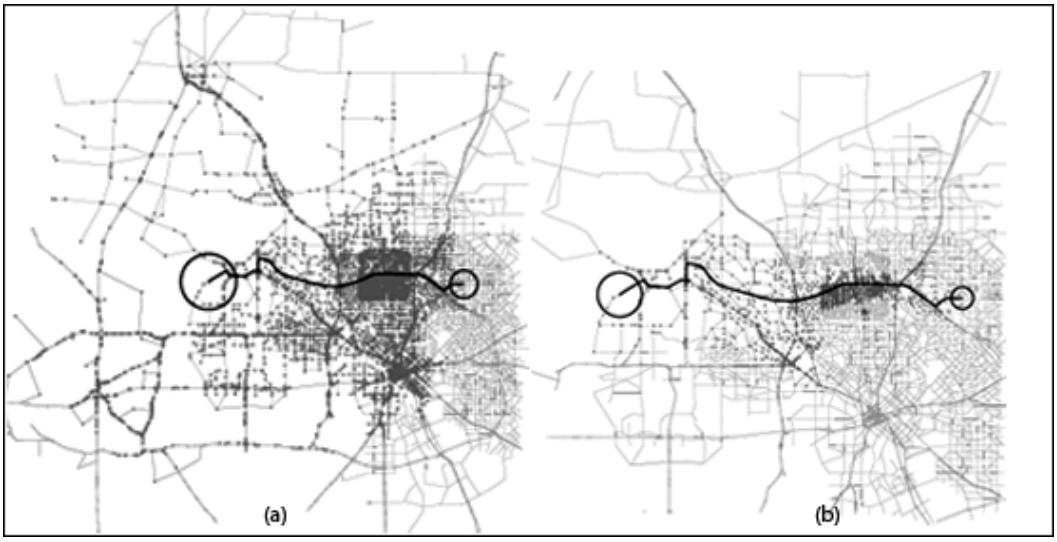


Figure 2.5: (a) Dijkstra Search, (b) A* Search on road network of Dallas Ft-Worth urban area

They have showed that A* Search is also much more efficient in terms of the number of nodes visited. Dijkstra's Algorithm searches much more nodes than A* does.

2.1.5 Human Navigation

There are also some studies to help human navigation, especially for visually impaired and deaf-visually impaired people. For example, Kulyukin, Gharpure, Nicholson and Pavithran [10] have used RFID technology and the modified potential fields algorithm for human navigation. They used a robotic guide, moving at moderate walking speeds, and it detects the free spaces around itself with sensors and laser range finders (Figure 2.6). They have placed RFID tags to the obstacles/objects in the environment. The guide detects these tags and finds its way between the objects. This approach is good for navigating in an unknown area like a campus, and finding the way without any need for a map database of that area. It also gives the administrator to change the place of the objects without changing any scene database.

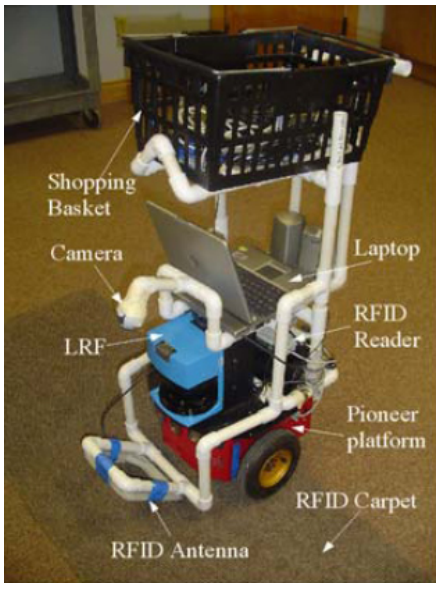


Figure 2.6: Kulyukin's Robotic Aid

Amemiya, Yamashita, Hirota and Hirose [11] have used RFID technology, motion sensor, a wearable computer and a wearable interface for Finger-Braille (Figure 2.7) in order to interact with deaf-visually impaired people. Test area was covered over 1300 RFID tags. When the user moves, system does the calculations and gives directions to the user by the vibration alerts on fingers.

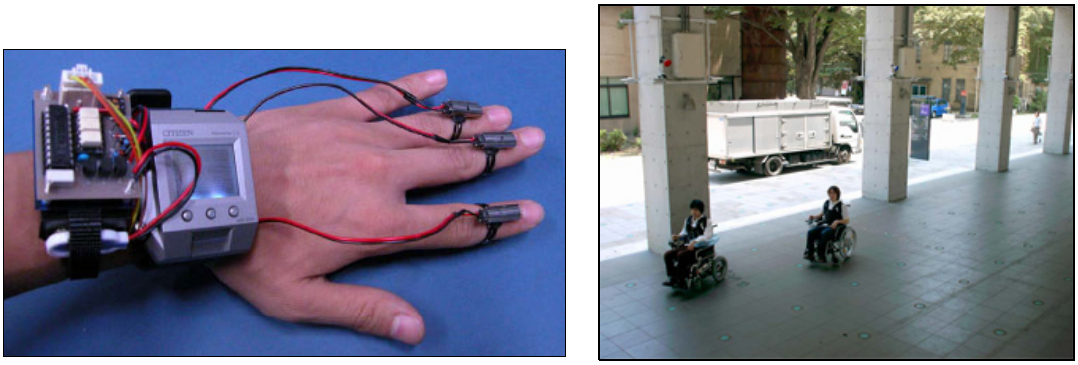


Figure 2.7: Wearable interface for Finger-Braille

Beside RFID technology, Global Positioning System (GPS) and ultrasound technology were also used in some studies. Ran, Helal and Moore [12] proposed a different system that can be used both outdoor and indoor. System works with Differential Global Positioning System (DGPS) at outdoor, and with ultrasound positioning system indoors (Figure 2.8). User can shift between these two systems

with a simple voice command. System can interact with the user as s/he moves and inform him/her about the environment. System switches to the ultrasound positioning system indoor, and uses simple geometry calculations to find the location.

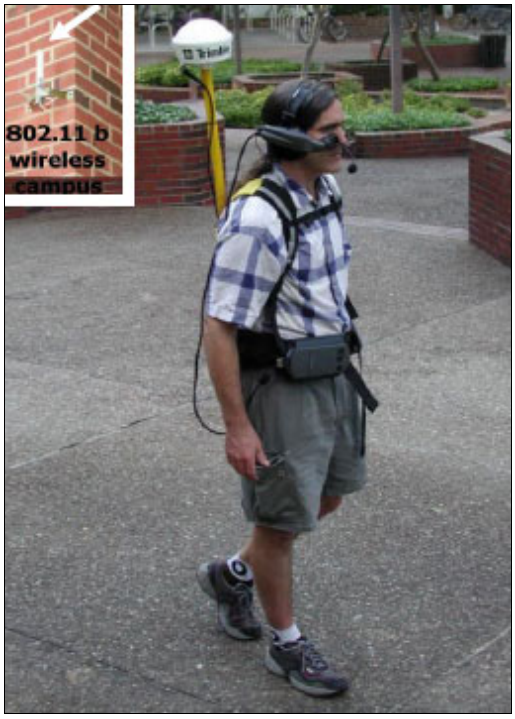


Figure 2.8: Drishti's navigation system

Generally, all of these systems are based on outdoor navigation designed to help the visually impaired people. These systems bring some advantages to users, and make their life easy, however, in order to make shopping available to these people we need to implement some of that technologies in one indoor system. We try to build a generic interface, so that it will be used easily in future studies as well.

In all of these systems user has to carry or ride some instruments which have big volumes. This brings extra load to the user. These systems should be improved in order for the integration of the visually impaired people to the real/normal world. Feasible systems should include small devices capable of doing all of the requested processes. They should be carried along without any problem. Hence, we prefer to place most of the processing units in a central system.

2.2. Path Finding Algorithms

There are many algorithms that are commonly known and used, ranging from simplex to complex, in order to be able to solve the path finding problem. The simplest approach is to walk directly towards the goal until met with any kind of obstacles. When met with any object, direction will be changed and it can be passed by tracing around the obstacle. This type of algorithm is an example of the "visually impaired Search" since it does not know or have any information about the path. Some other examples of the visually impaired Search are Breadth-First Search, Dijkstra's Algorithm and Depth-First Search.

There also exist some algorithms that plan the whole path before moving anywhere. Best-first algorithm expands nodes based on a heuristic estimate of the cost to the goal. Nodes, which are estimated to give the best cost, are expanded first. The most commonly used algorithm is A* algorithm, which is a combination of the Dijkstra algorithm and the best-first algorithm.

The different algorithms work in different ways. The breadth-first search begins at the start node, and then examines all nodes one step away (here we have 4 different nodes for Manhattan distance calculation and 8 different nodes for Euclidean type of distance calculation), then all nodes two steps away, then three steps, and so on, until the goal node is found. This algorithm is guaranteed to find a shortest path as long as all nodes have a uniform cost.

The bi-directional breadth-first search is where two breadth-first searches are started simultaneously, one at the start and one at the goal, and they keep searching until there is a node that both searches have examined. The final path is the combination of the path from the start to the intersection node, and the path from the goal to the intersection node.

2.2.1 Dijkstra's algorithm

Dijkstra's algorithm (named after its developer, E. Dijkstra [13]) looks at the

unprocessed neighbors of the node closest to the start, and sets or updates their distances (in terms of cost, not number of nodes) from the start. The Dijkstra algorithm expands the node that is farthest from the start node, so it ends up "stumbling" into the goal node. Just like the breadth-first search; it is guaranteed to find the shortest path (Figure 2.9).

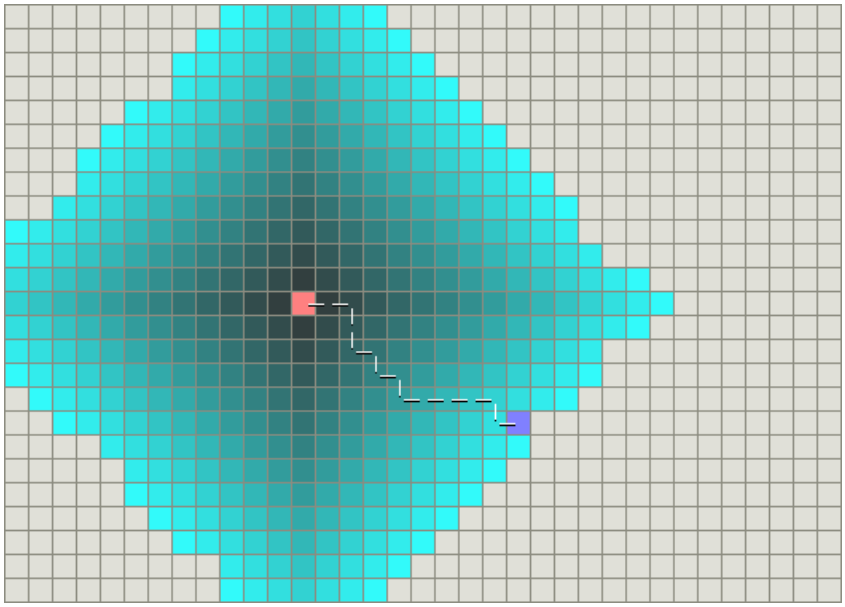


Figure 2.9: Dijkstra's Search (from Amit Patel's Game Programming web site [14])

The depth-first search extends nodes (it extends a node's descendants before its siblings) until it either reaches the goal or a certain cut-off point, it then goes onto the next possible path .

The best-first search algorithm is a heuristic search algorithm, meaning that it can take knowledge about the map into account. It is similar to Dijkstra's algorithm, but it goes to the node closest to the goal, rather than the node furthest from the start.

2.2.2 A* Algorithm

The algorithm was first described in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael [**15**]. It is a global space-search algorithm that can be used to find solutions to many problems, path finding being just one of them. It has been used in many

real-time strategy games and is probably the most popular path finding algorithm.

The A* algorithm works much like the Dijkstra and best-first algorithms only it gives values to the nodes in a different way. Each node's value is the sum of the actual cost to that node from the start and the heuristic estimate of the remaining cost from the node to the goal. In this way it combines the tracking of previous length from Dijkstra's algorithm with the heuristic estimate of the remaining path from the best-first search. In fact, the Dijkstra search is an A* search, where the heuristic is always 0. This algorithm also makes the most efficient use of the heuristic function, meaning that no other algorithm using the same heuristic will expand fewer nodes and find an optimal path.

A* algorithm uses a starting point and a destination point to produce the desired path, if it exists (Figure 2.10). In figures, the cell marked with "O" is our starting point/node and the cell marked with "X" is the destination. White squares are walkable nodes and the black ones are walls, shelves or any other obstacles.

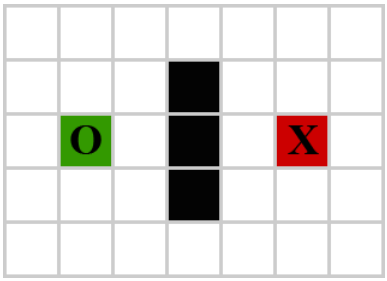


Figure 2.10: Starting and Destination Points in A* Search Algorithm

A* algorithm starts to execute by looking at the starting node first and then expanding to the surrounding nodes (Figure 2.11).

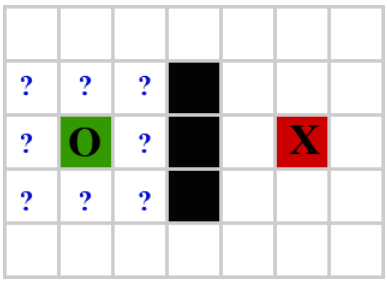


Figure 2.11: Starting the A* Search Algorithm

This operation continues until the destination node is found. In order to know the nodes, which will be used in search, A* algorithm needs a way to keep track of the nodes. So the nodes to be examined are held in a list, called Open List. At the beginning we place the starting node to the Open List, and after examining all of its surrounding nodes we will move it from the Open List and place in another list called the Closed List (Figure 2.12). Closed List holds the nodes that are visited and there is no need to re-visit its members. When building the Open List, algorithm checks if the node is walkable. If the node is not walkable, it is not added to the Open List.

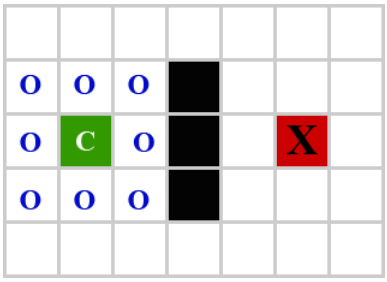


Figure 2.12: Putting the neighboring cells to the Open List

This process made for one cell is the main iteration through one A* loop; however, we need to track some additional information. We need to know how the nodes are linked together. Although the Open List maintains a list of adjacent nodes, we need to know how the adjacent nodes link together as well. We can do this by tracking the parent node of each node in the Open List. A node's parent is the single node that the user steps from to get to its current location. On the first iteration through the loop, each node will point to the starting node as its parent (Figure 2.13).

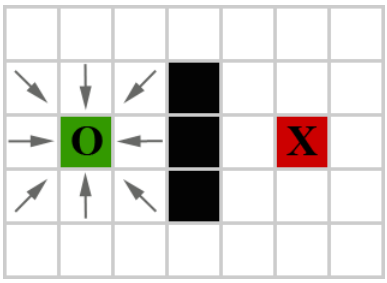


Figure 2.13: Parent relation of Starting Node

We will use the parent links to trace a path back to the starting node when we reach the destination. At this point we start over process. We now have to choose a new node to check from the Open List. At the first iteration we had only a single node in the Open List. We now have eight nodes in the Open List, and the node which will first be inspected is determined by assigning a score to each node. This score, f(n), is the combination of two scores:

$$f(n) = g(n) + h(n)$$

where g(n) is the cost of the path from the starting node to any node n,

h(n) is the heuristic estimated cost from any node n to the goal.

We select the node with the lowest score. We use a Priority Queue to build the Open List, and the nodes that will be added to Open List is sorted by this score. So when we pop an element from this Queue, we always get the node with the lowest score. There are some well-known heuristics used in scoring.

2.2.2.1 Heuristic Functions Used in A* Algorithm

We calculate each node's score by adding the cost of getting there from the starting location to the heuristic value, which is an estimate of the cost of getting from the given node to the final destination. We use this score when determining which tile to check next from the Open List. We will first check the tiles with the lowest cost. In this case, a lower cost will equate to a shorter path.

The g(n) value shown in each open node is the cost of getting there from the starting node. In this case, each value is 1 because each node is just one step from the starting node. The h(n) value is the heuristic. The heuristic is an estimate of the number of

steps from the given node to the destination node. We do not take obstacles into consideration when determining the heuristic. We have not examined the nodes between the current node and the destination node, so we do not really know yet if they contain any obstacles. At this point we simply want to determine the cost, assuming that there are no obstacles. The final value is f(n), which is the sum of g(n) and h(n). This is the cost of the node. It represents the known cost of getting there from the starting point and an estimate of the remaining cost to get to the destination.

Heuristics used in A* algorithm have some variations, but for grid based maps there are three heuristic functions that work well.

A. Manhattan Distance:

$$h(n) = D * (|n.x - goal.x| + |n.y - goal.y|)$$

where D represents the cost of moving one node to one of its neighbors

Manhattan Distance (Figure 2.14) allows us to move in four directions (North, South, East and West).

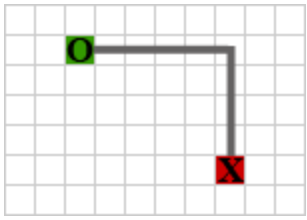


Figure 2.14: Manhattan distance calculation

The Manhattan heuristic is computed by adding the differences in the x and y components together. The advantage of using this heuristic is that, it is computationally inexpensive, and it can run faster than the others. The major disadvantage of the Manhattan heuristic is that it tends to overestimate the actual minimum cost to the goal (unless 4-adjacency is used) which means that the road being found may not be an optimal solution. If we are not interested in an optimal solution, but just a good one, then using an overestimating heuristic can speed up the road finding.

B. Diagonal Distance:

$$h(n) = D * max(|n.x - goal.x|, |n.y - goal.y|)$$

Diagonal Distance allows us to move in 8 directions (Figure 2.15).

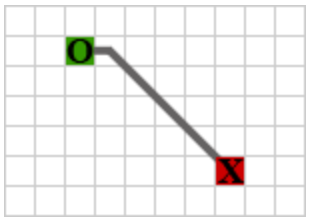


Figure 2.15: Diagonal distance calculation

Diagonal method balances the h(n) and g(n) in calculation of the total cost, but it is a bit slower than Manhattan method.

C. Euclidean Distance:

$$h(n) = D * \sqrt{((n.x - goal.x)^2 + (n.y - goal.y)^2)}$$

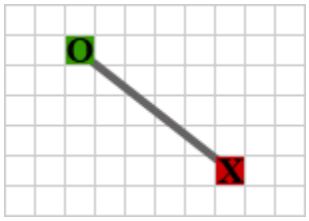


Figure 2.16: Euclidean distance calculation

The Euclidean heuristic (Figure 2.16) is admissible, but usually underestimates the actual cost by a significant amount. This means that we may visit too many nodes unnecessarily which, as a result, increases the time it takes to find the road. The Euclidean distance heuristic is also computationally more expensive to apply compared to the Manhattan heuristic, as it additionally involves two multiplication operations and calculating the square root.

As we search the nodes toward the goal we add them to Open List and transfer to

Closed List when our job with them completed. These operations continue until we reach to the goal.

When we reach to the destination, we do a back-track by using the parent links for all of the nodes and construct the path (Figure 2.17).

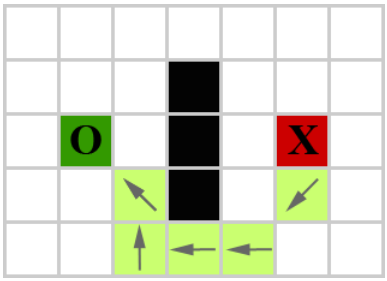


Figure 2.17: Constructing Final Path in A* Search

2.2.3 Pseudo Code for A* algorithm

Create a node containing the goal state node_goal

Create a node containing the start state node_start

Put node_start on the open list

while the OPEN list is not empty {

Get the node off the open list with the lowest f and call it node_current

if node_current is the same state as node_goal we have found the solution;

break from the while loop

Generate each state node_successor that can come after node_current

for each node_successor of node_current {

Set the cost of node_successor to be the cost of node_current plus the cost

to get to node_successor from node_current

find node_successor on the OPEN list

if node_successor is on the OPEN list but the existing one is as good or better

then discard this successor and continue

if node_successor is on the CLOSED list but the existing one is as good or better

then discard this successor and continue

Remove occurences of node_successor from OPEN and CLOSED

Set the parent of node_successor to node_current

Set h to be the estimated distance to node_goal (Using the heuristic function)

Add node_successor to the OPEN list

}

Add node_current to the CLOSED list

}

CHAPTER 3

Research Methodology

This study mainly concentrates on developing a software utility that will assist to solve the problem of visually impaired people. As a case study, the proposed solution implements a path finding algorithm to a prototype application whose demonstration area will be a simulated shopping store. Our software will interact with some outer systems, like a Hand Held Navigation Unit (HHNU) that will be given to the user, RFID readers connected to the computer via RS232 serial port and database server.

Hand held unit measures the Radio Signal Strength values from the 3 RFID beacons as power density. After that it transmits these 3 power values to the Central Processing System by using a transmitter (RFID tag). In the computer, there is an RFID reader that reads the transmitted parameter. Our software has a connection with this reader using RS232 communication port. All the values collected from this port are in power values, so we have to find their distance equivalents after some formulation and process. After that we use the trilateration method to find the intersection point of these 3 distances.

After finding the location of the user, we need the destination point which we get from the dial pad with user's selection. After we get the location and the destination information, we can compute the shortest path between these two points. Since we hold the available and unavailable locations/cells in the memory, we can easily calculate the path by using any path finding algorithm. There are various algorithms capable of doing this calculation, but the best candidate is A* Algorithm. As discussed in the literature review, different studies showed that it is more suitable to implement and the fastest algorithm when used in a known area.

In this study, found path is sent to the hand held unit in order to allow the user to follow the found path. A special compass, developed for the visually impaired people, guides the user when s/he moves. Our system also checks the location of the user all time in order to know his/her location, so that we can detect any movement errors. In case of going to the wrong directions accidentally, we always monitor the user and correct his/her path if it is necessary.

Lots of studies on visually impaired navigation or help aids for visually impaired people have been done before. Some of them had tried the same technologies and algorithms, and we tried to implement it to a prototype model. We have built a setup to simulate the shopping store in lab environment and we have debugged and improved our system by doing some experiments.

Our software also has a simulation option to go offline, by selecting initial and final points and calculating the path. In offline mode it is possible to compare different algorithms and cost calculation methods.

In this study, first the “PUSULAL ADMIN” software is developed by using Java language. We selected Java since it is a very high level and powerful programming language, and it can run on different Operating Systems (OS). It can find the path according to two points, start and destination, but we cannot guarantee that the user follows the exact path we directed his/her to. So we have to check the user's position at all times. Hence, as a second step we implemented a Statistical Localization method which reads the RFID values and uses Bayes Decision Theory to check the received RFID signals with the previously known values for the neighboring cells. We can modify the path in any case of dispersion from the sent path.

Next we integrated our software to the other parts of the PUSULAL system. We got some measurements at the serial port with RFID transceiver.

We also compared different path calculation methods (Manhattan, Euclidean and

Diagonal) in path finding algorithm, both in the feasibility of the found path and the calculation time.

CHAPTER 4

Simulation Results and Discussions of the Pusulal Admin

4.1 Pusulal Interface

Pusulal interface is developed in Java language by using Netbeans IDE (version 5.5). Netbeans is supported by Sun Microsystems, developer of the Java language. It is a clear and powerful development environment. Netbeans can be downloaded from "www.netbeans.org" web site.

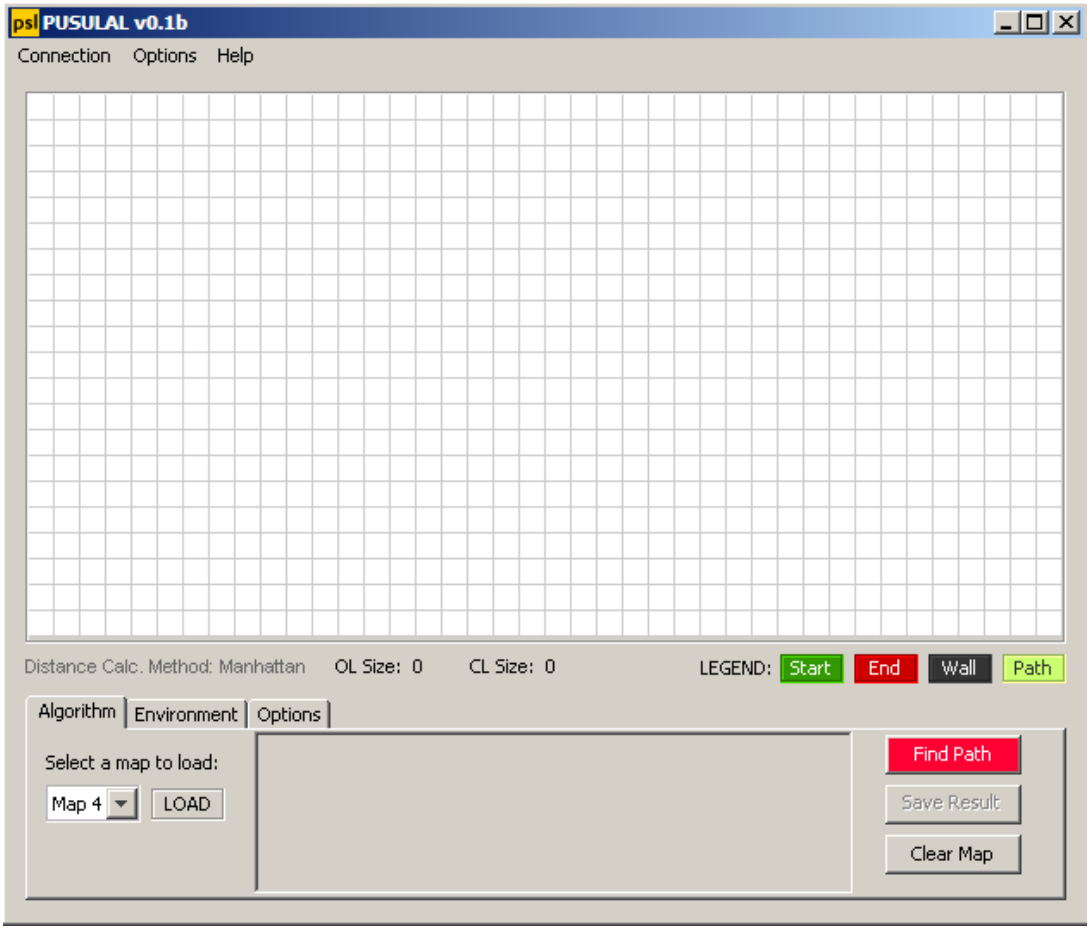


Figure 4.1: PUSULAL Interface

We used Spiral Modeling as a Software Development Methodology while developing the PUSULAL interface, which is seen on Figure 4.1. It has an area simulating the shopping center, viewed from top and divided by grids. Under that area there exist some information messages related with Distance Calculation method for Heuristic estimation and a Legend part showing the meanings of the colors used in simulation area (Figure 4.2). It is also possible to change the grid area by selecting the appropriate element from the Legend and then clicking on the cells of the map.

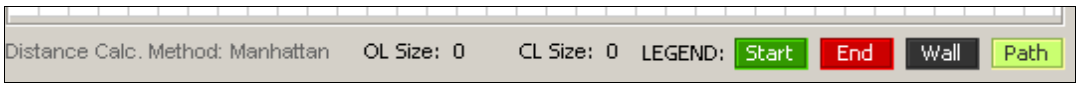


Figure 4.2: PUSULAL Interface Legend and Info Part

Under the Legend bar there is an option and operation area; which consists of a Map Load section to load some predefined maps from memory, a text area to inform the user about the process and founded path and some buttons to operate the algorithm (Figure 4.3).

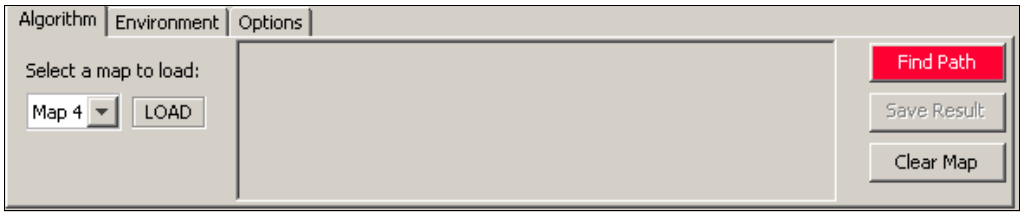


Figure 4.3: PUSULAL Interface Option and Operation Part

By selecting a map and pressing the Load button, it is possible to load the selected map to the grid area (Figure 4.4). If we do not want to load a predefined map and prepare our own map, it is possible to select "Wall" option from Legend Part. Then, obstacles can be placed by clicking the blank cells in the grid area. After completing the map, we can use Start and End buttons in the Legend. By default, first click on the map makes the cell a "Start" node, the second one an "End" node, and other clicks make the cells non-walkable nodes (black color).

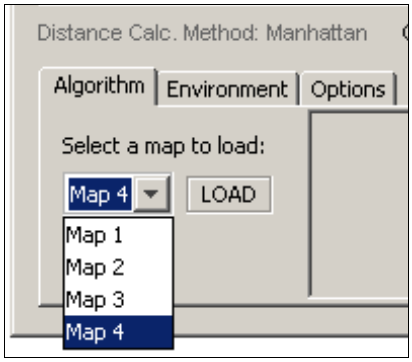


Figure 4.4: Selecting a map to load

Our system consists of some menu items that can be summarized in Table 4.1:

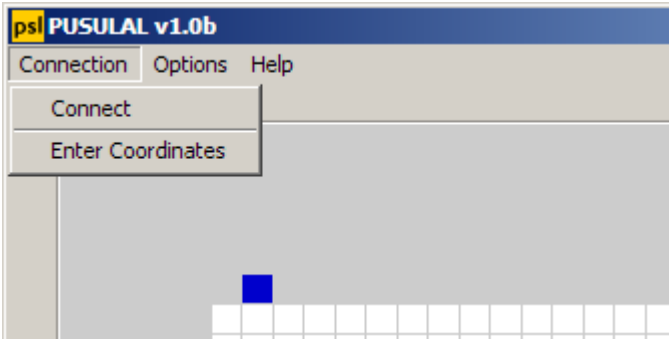
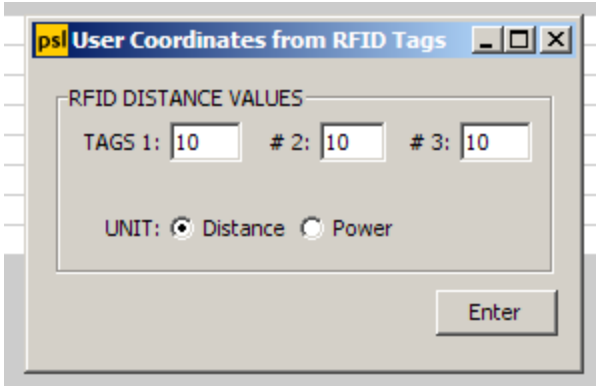
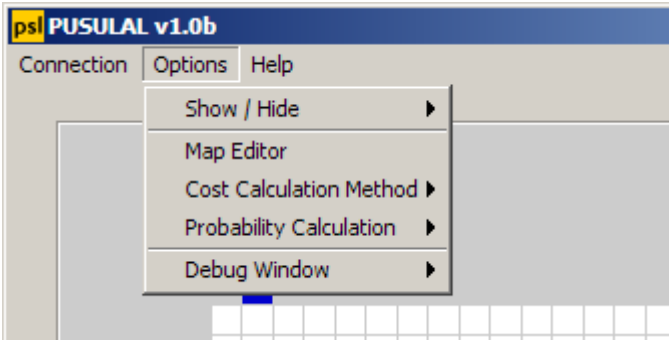
 **(a)**	Connection Menu Item makes it possible to work online or offline. When we select to work online it tries to find and then listen the serial ports. In offline mode we can simulate different parameters.
 **(b)**	When worked offline, the RFID values can be entered through the keyboard. This feature is added for simulation purpose. We are using the same location finding algorithm in both online and offline modes.
 **(c)**	In Options menu, there exists some parameters that can be set by the system moderator.

Table 4.1: Menu Items in PUSULAL Admin Software

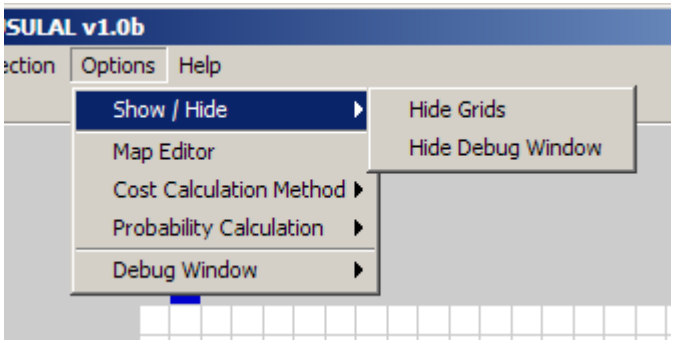
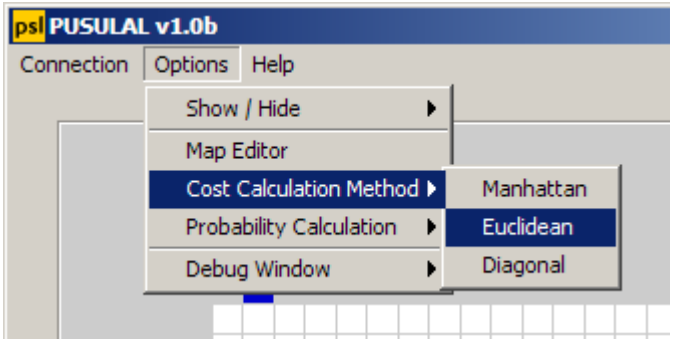
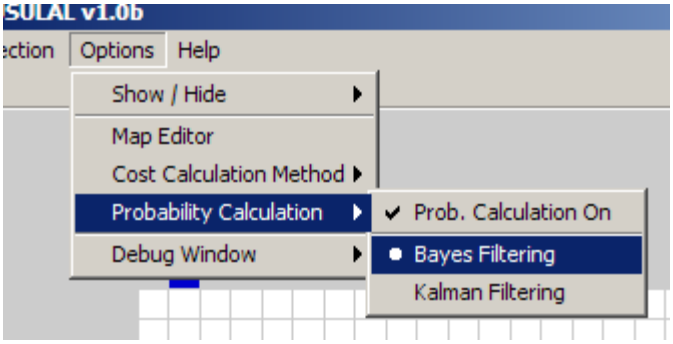
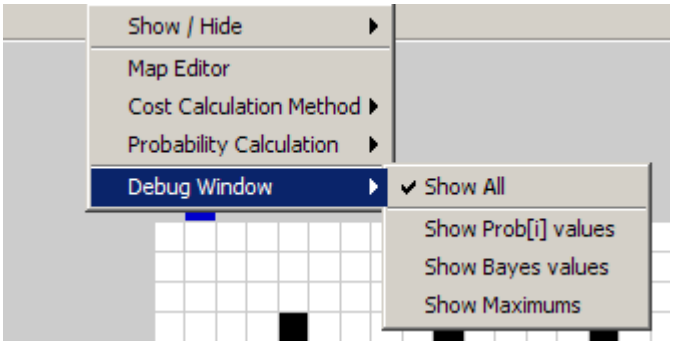
 **(d)**	We can show or hide the Grids on the Table that we use to simulate the area. Or we can show / hide the Debug window for various purposes.
 **(e)**	We can change the cost calculation method used in path finding process in A* Algorithm. It can be in 3 different types: - Manhattan - Euclidean - Diagonal
 **(f)**	We can also set the Probability Calculation function to "On" or "Off". If it is set to "On", then it is possible to select the type of Filtering. Bayes and Kalman, which can be implemented later, can be used.
 **(g)**	It is possible to see all of the variables for every inspected cell in Probability Calculation function. This can be useful for tracking a bug or problem.

Table 4.1 cont. : Menu Items in PUSULAL Admin Software

Our system is ready to select the Start and End nodes after loading a map or making a different map by clicking,. This can be done by clicking on the map if the system is not online or not connected to the RFID system. In Figure 4.5, a sample shopping center is simulated with shelves placed in order.

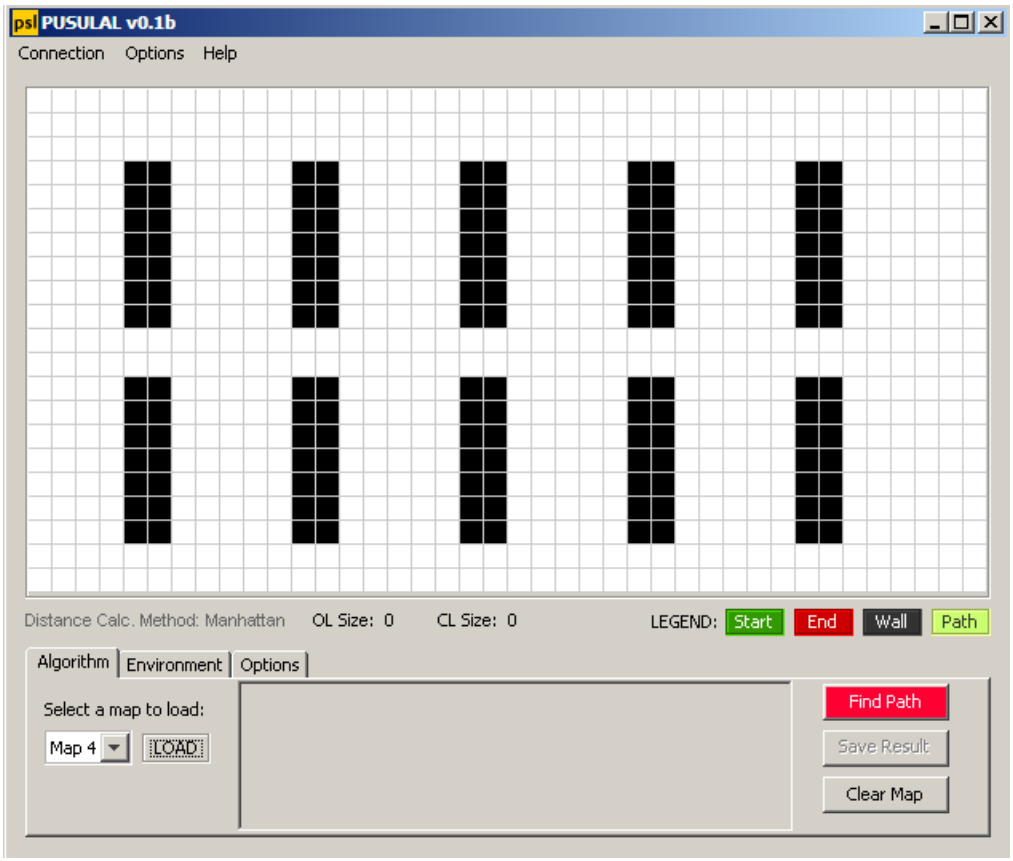


Figure 4.5: Simulation of a shopping center in PUSULAL Admin Interface

Then we select the Start and End points, which can be seen in Figure 4.6.

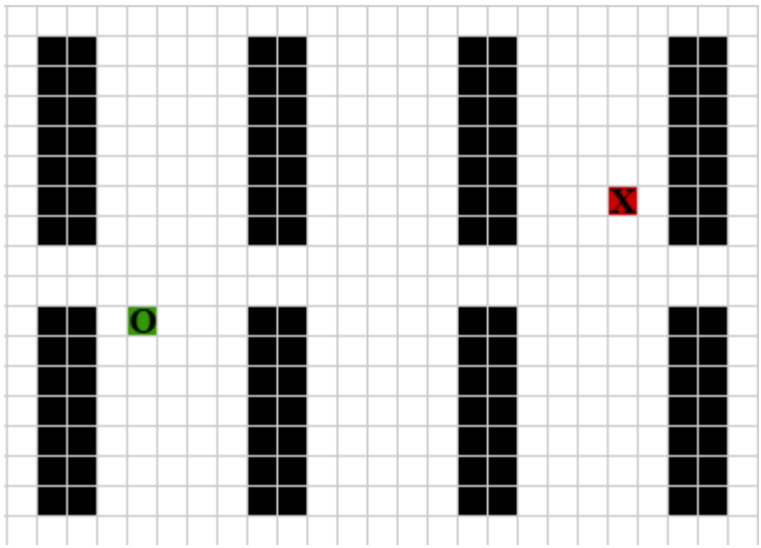


Figure 4.6: Selecting the Start and End points in the Map

After selection we can start to search the map by clicking the red “Find Path” button. When the system finds a path it is showed in dark gray color on the screen (Figure 4.7). The light gray cells are examined cells.

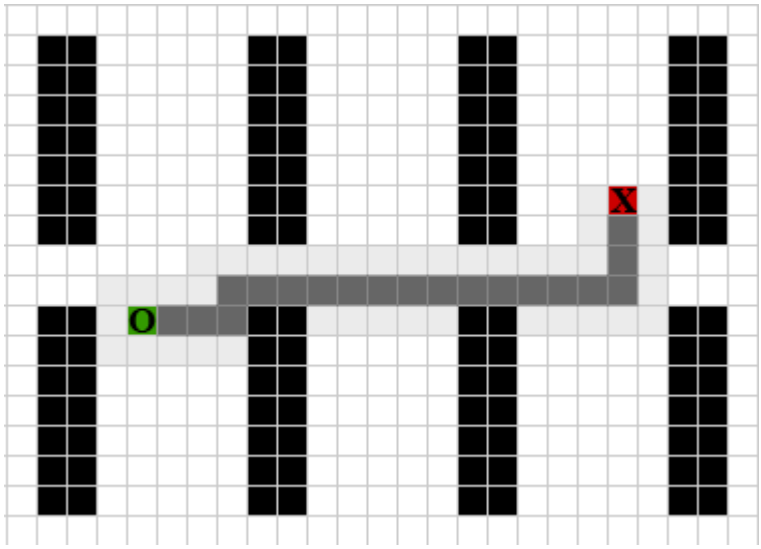


Figure 4.7: Found path in Manhattan method

Distance calculation method is set to Manhattan method by default. If we want to see the path with Euclidean method, we can change it from the Options menu (Options -> Cost Calculation Method -> Euclidean) (Figure 4.8).

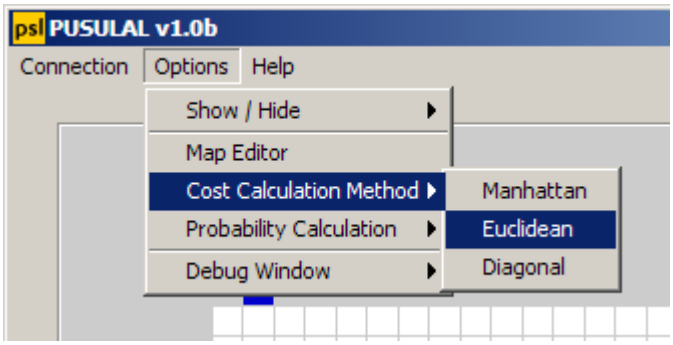


Figure 4.8: Changing Cost Calculation Method.

To find the new path with changed Cost Calculation Method. We will just click the Find Path button again. New path will be shown on the map (Figure 4.9). Also we can follow the cells and information in the Option and Operation Part (Figure 4.10).

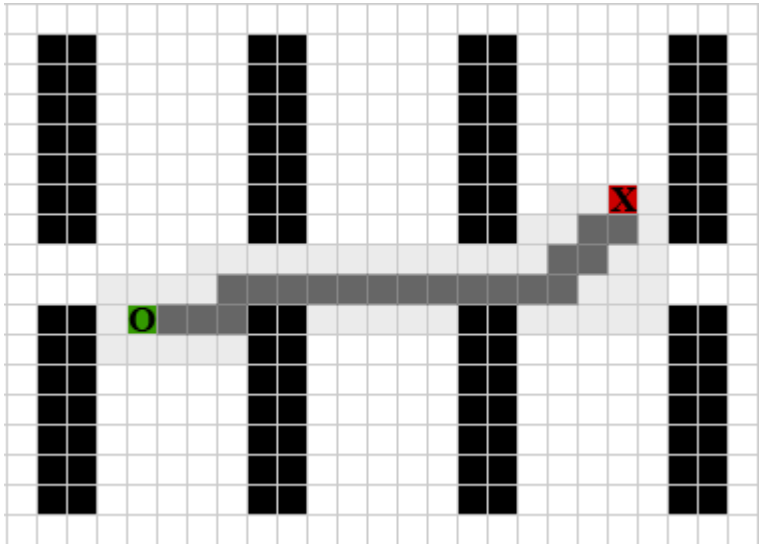


Figure 4.9: Found path in Euclidean Method

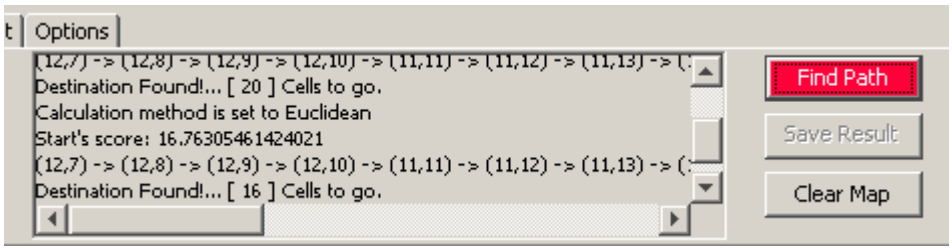


Figure 4.10: Path info on Option and Operation Part (OaO Part)

Our system can find the path, but this is the path which we assume that user will always follow. However, in real cases the situation can be different from that. We can find a path from one Starting point and one Destination point, and then send it to the user. But we cannot be sure that user is on the cell that we have directed him/her to. Since we always get RFID signals from the HHNU, we can estimate the position of the user. Because of the errors in measurements due to the resolution and due to scattering, reflections and refractions of the radio signals of the RFID devices we used, we get similar signal strength values from neighboring cells. For this reason we integrated a path prediction system to our utility. We will go into detail of this system in the next section.

4.2 Location Prediction

In real RFID measurements, the used experiment area is constructed in the Advanced Electronics Laboratory (Figure 4.11). Measurement area is divided into 60cmx60cm cells and group of four cells are coded with the characters.

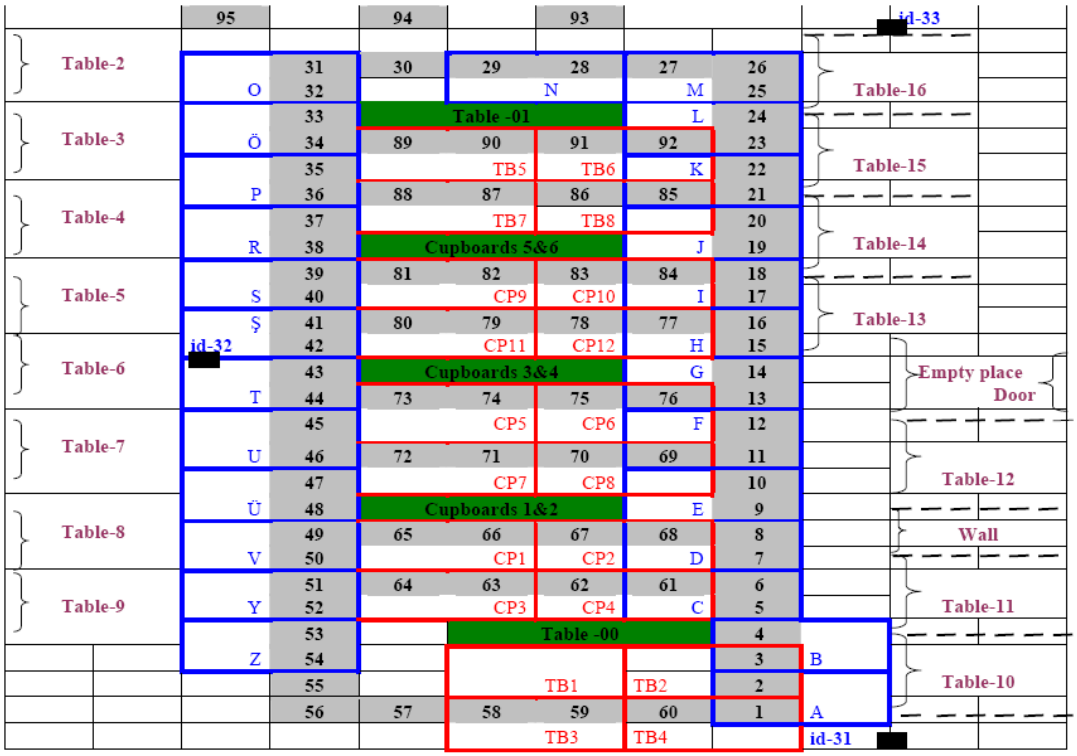


Figure 4.11: Experimental area used for collecting the Radio Signal Strength values

There are 46 cells (A-Z, TB1-TB8 and CP1-CP12) and in every cell 52 RFID measurements were taken. To simulate this experiment area, our program is prepared as in Figure 4.12.

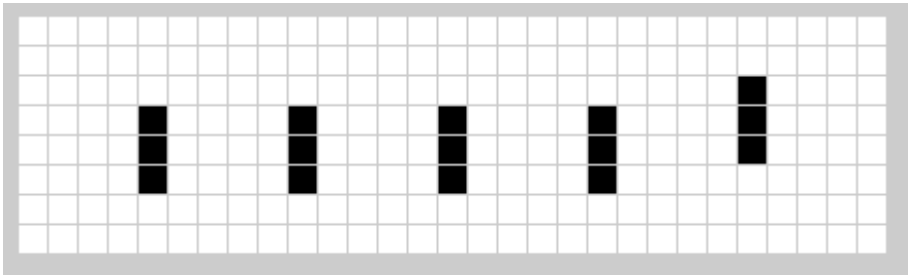


Figure 4.12: Experiment Area Simulation in PUSULAL

When we load the first map from the memory, we create the similar environment in our program. Figure 4.13 shows the cells of the simulated area better. To avoid coincidence in the RSS values some cells are modified, since there is some overlapping group cells in the experiment area (like A and TB4, B and TB2,...).

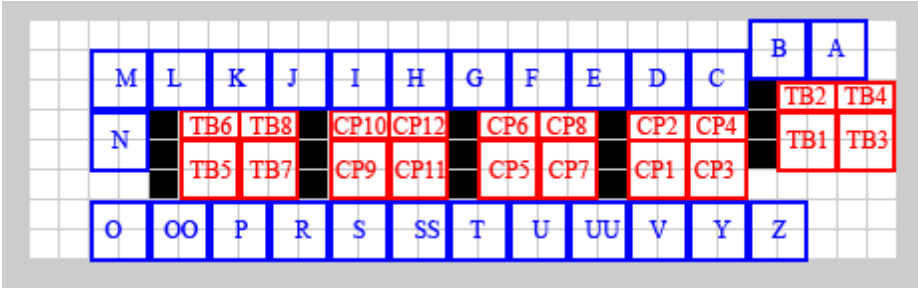


Figure 4.13: Experiment Area showed in detail in PUSULAL Interface

Every character coded cell has its own Radio Signal Strength (RSS) values and collected in text files, named the same with the code. These files have 52 different measurements from 3 different RFID beacons. Example of the RSS values can be seen on Appendix D and E.

These numbers are used in the calculation of probability of being on the redirected path. There are two filters commonly used in the probability calculation: Bayes Filtering and Kalman Filtering. We used Bayes Filtering at this moment, but it is possible to add the second one to our system. It is possible to turn on/off the probability calculation from the Options Menu (Figure 4.14). If Probability calculation is set to “On”, then we can select the appropriate filter to run. Otherwise, we select “Off”, then filter selection is also deactivated.

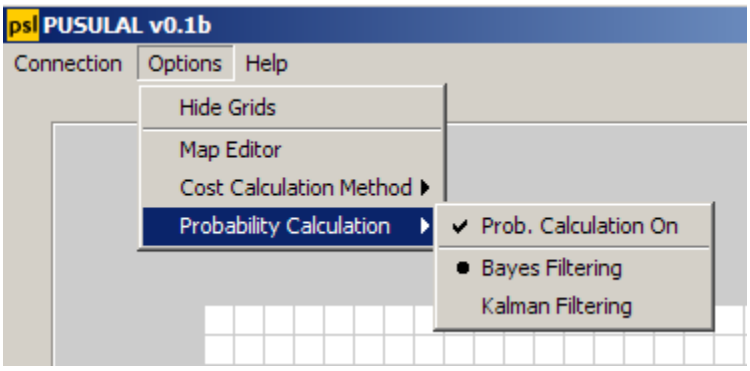


Figure 4.14: Changing Probability Calculation settings

Bayes Filtering uses some statistical methods to predict the next steps in a given value, by using the RSS values and a given probability constant for each cell [16]. Adapted formula of Bayes Filtering for our system is given in Figure 4.15.

$$\sum \left[\log\left(1/\sqrt{(2*\pi*\text{variances}[i])}\right) - (1/2)*(\text{currentRSS}[j]-\text{means}[i])^2 / \text{variances}[i]) \right] + \log(\text{prob}[i])$$

Figure 4.15: Adapted Bayes Formula

Where:

variances[i] is equal to the variance value of the *i*th cell

currentRSS[j] is the RSS value of the unknown cell that we read from RS232 port

means[i] is the mean value for the cell i

prob(i) is the transition probability of the neighboring cells that we distribute.

When "Probability Calculation" is set to "On", we can see its result in the debug window (Figure 4.16). When examining any cell for Bayes method, we assign some constants to the neighboring cells. We assume that being on the cell that we guide with A* path has the highest probability, second highest probability for being on the same cell and least probability for being on other cells.

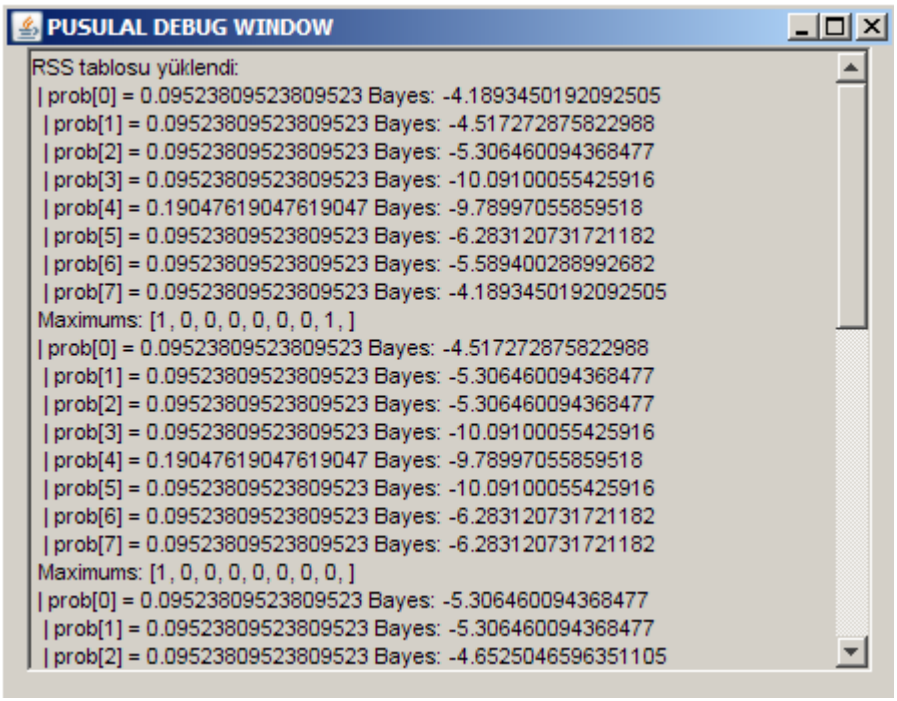


Figure 4.16: Debug Window

According to the Bayes calculation we get a maximum values array "*Maximums[]*", indicating that our user is most probable on these cells, with the help of measured RSS, their mean/variance values and probability parameters. Each index in the *Maximums[]* array shows us the neighboring cells as in Figure 4.17.

Figure 4.17: Numbering of Neighbouring cells

Each time we run the Find Path command, we can see the *Maximums* array values in the Debug Window (Figure 4.18). Calculation sample is done above by using only one RSS value from the RSS table. In order to get a more reliable and rich result we have to use all of the 52 measurement values for all neighboring cells. In this case we get better results, however in our experiment setup since we gave the same RSS

values to more than one cell we are facing with more than one maximums in the same neighboring. It can be solved by giving different values to each separate cell.

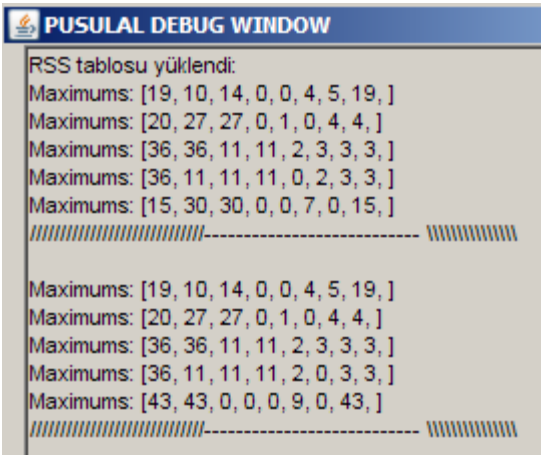


Figure 4.18: Probability Calculation results in Debug Window

However, on the other hand, we experienced some problems with the simulation of the experiment area to our utility, since the experimental classification of the real measurements is based on one big cell, such as A, which consists of 4 small cells (Figure 4.13). These small cells have equal RSS values, resulting in the same Mean and Variances values, which causes problems for selecting the next step cell in the Bayesian Prediction. Since the neighboring cells have the same values algorithm cannot calculate the path properly. It creates more than one maximum value for one of the cells in the same neighborhood in the calculation of Bayes values.

One alternative was to arrange our system to exactly match with the experimental space, but in this case we can lose some precision because of the dimensions of the cells. Each small cell has 60cm X 60cm dimensions, but the big cells are 120cm X 120cm in size.

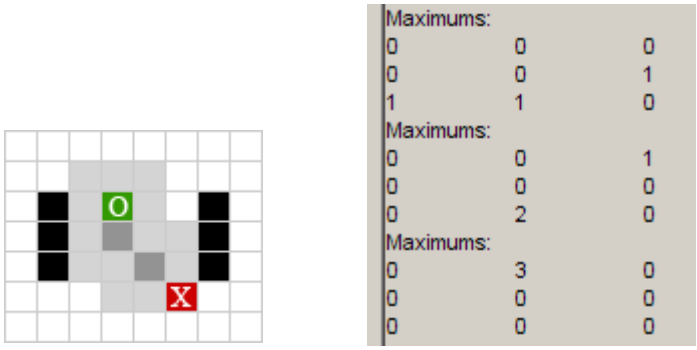
We preferred the second alternative; distributing the RSS values to each individual cell. We distributed 20 random RSS values to each small cell in one big group. This size is enough to make statistical analysis such as Variance, although larger number of data gives better result. We divide each big group cells (A-Z, CP1-CP12, TB1-

TB8), total of 46, to 164 small cells (plus 7 extra cells for uncovered regions, total of 171) (Figure 4.19).

																				5	6	1	2	
49	50	45	46	41	42	37	38	33	34	29	30	25	26	21	22	17	18	13	14	9	10	7	8	3	4	
51	52	47	48	43	44	39	40	35	36	31	32	27	28	23	24	19	20	15	16	11	12		107	108	109	110
53	54		59	60	61	62		71	72	73	74		83	84	85	86		95	96	97	98		111	112	115	116
55	56		63	64	67	68		75	76	79	80		87	88	91	92		99	100	103	104		113	114	117	118
57	58		65	66	69	70		77	78	81	82		89	90	93	94		101	102	105	106	167	168	169	170	171
119	120	123	124	127	128	131	132	135	136	139	140	143	144	147	148	151	152	155	156	159	160	163	164			
121	122	125	126	129	130	133	134	137	138	141	142	145	146	149	150	153	154	157	158	161	162	165	166			

Figure 4.19 Modified Simulation Area for Probability Calculation

After calculating the Mean and Variance values and saving for the text files, we can use this new values. In Figure 4.20, we can find the calculated path with A* Algorithm (a) and calculated path according to the Bayes (b). Now we can follow the two paths simultaneously.



(a) (b)

Figure 4.20 (a) Proposed path by A* Algorithm, (b) Estimated Path by Bayes Classification

Later we changed the neighbouring cells from 9 to 25 in order to check the 2nd order neighbours of the cell that is being on the test. It can be changed from Menu as Options -> Neighbourhood Number (Figure 4.21).

If we change the Neighbourhood Number in Bayes analysis, we can now see the 2nd order neighbours in Debug Window (Figure 4.23). In this case numbering of these

cells is as in Figure 4.22.

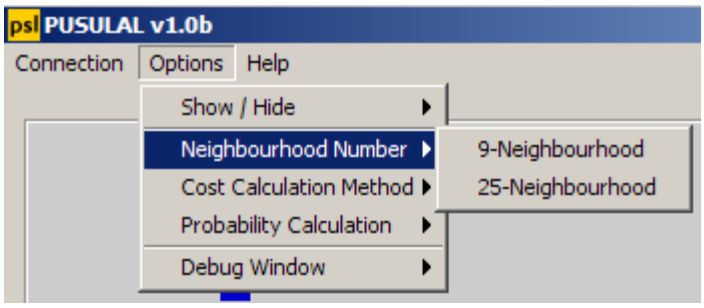


Figure 4.21: Changing Neighbourhood Number for Bayes from Options

9	24	23	22	21
10	0	7	6	20
11	1	8	5	19
12	2	3	4	18
13	14	15	16	17

Figure 4.22 Numbering of Cells in 25-Neighbourhood

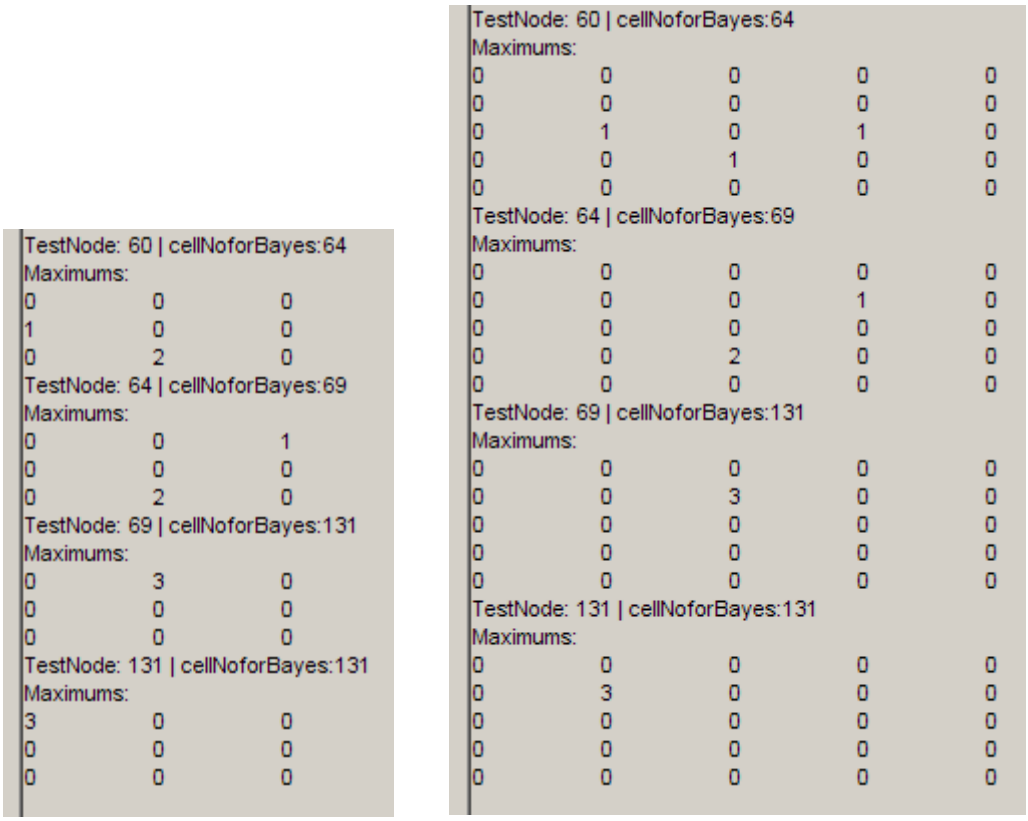


Figure 4.23 (a) 9- and (b) 25-Neighbourhood for Bayes analysis

We can see the differences in paths between A* and Bayes Method in Figure 4.24 and Figure 4.25. The cells filled with light green color is the path found with A* Algorithm and the cells with dark green framed builds the path from Bayes method.

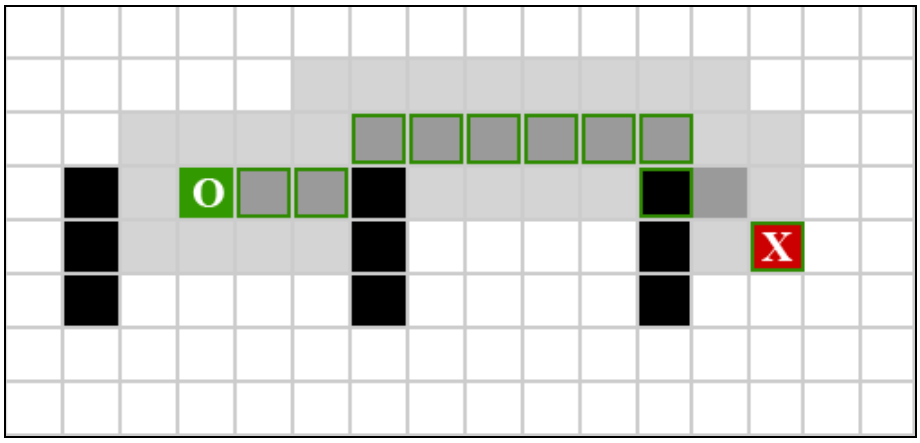


Figure 4.24 Differences in found path between A* Algorithm and Bayes -1

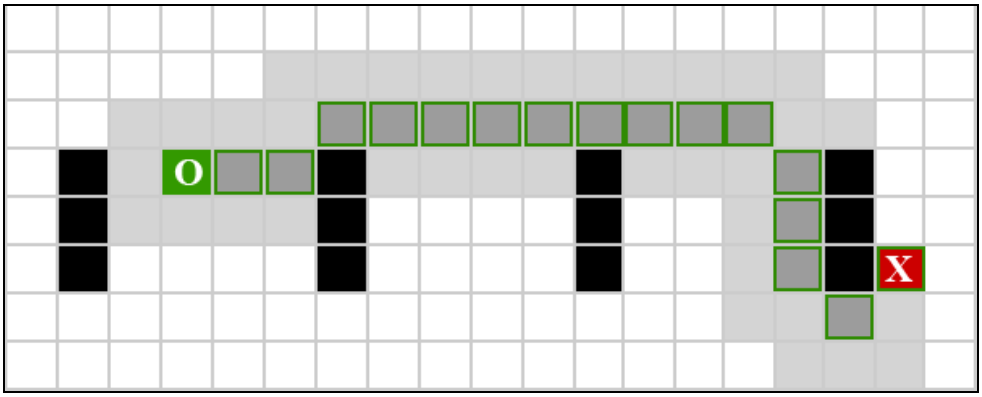


Figure 4.25 Differences in found path between A* Algorithm and Bayes -2

4.3 Integration to the Other PUSULAL Subsystems

4.3.1 Serial Port Connection to the RFID Tag

We are getting the real time RFID values from the beacons by using the RS232 Serial Port of the computer. One RFID transceiver is connected to the serial port and it reads the RFID Radio Signal Strength values from the 3 RFID beacons and transfers them to the PC. Our software reads them by using an extension to the standard Java library, called "javax.comm package". Data coming from the transceiver at the serial port can be seen in Table 4.2:

Table 4.2: Data sample from the RFID transceiver

Found port: COM9
61 61 0d 0e 41 0e 0c 0b 81 0d 54 54
61 61 0c 0b 41 0c 0d 0c 7d 0d 54 54
61 61 0c 0d 41 0b 0d 0a 7c 0d 54 54
61 61 0d 0b 40 0c 0c 0f 7f 0d 54 54

Data from transceiver comes in hexadecimal format, and a conversion is applied to it to change its format to decimal. The first 3 numbers after the "61 61" group corresponds to the RFID signal strength values from 3 beacons, id31, id32 and id33. Data comes with these order also, first one is from id31, second one from id32 and third one from id33.

For example,

for the line 61 61 0d 0e 41 0e 0c 0b 81 0d 54 54
0d from id31 (13 in decimal)
0e from id32 (14 in decimal)
41 from id33 (65 in decimal)

If the software is set to work offline, we can select the start and destination points by

mouse or, for start point it is possible to enter the RSS or distance values with the keyboard (Figure 4.26). Distance or power (RSS) values can be entered by selecting appropriate units.

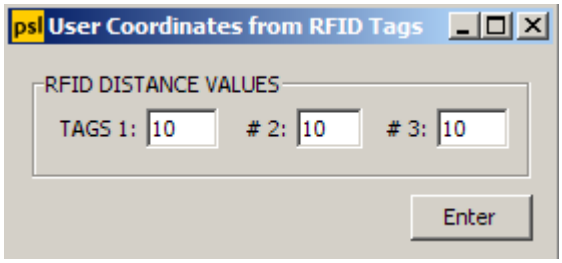


Figure 4.26: Entering the RFID RSS values from keyboard

4.3.2 Location Finding / Calculation

After reading the RFID values from the beacons, we can compute the location of the HHNU. If we know the location of 3 reference points we can compute the unknown location in 2D position by using the Trilateration technique [16]. Our reference points are the RFID beacons we placed to the laboratory (Figure 4.27).

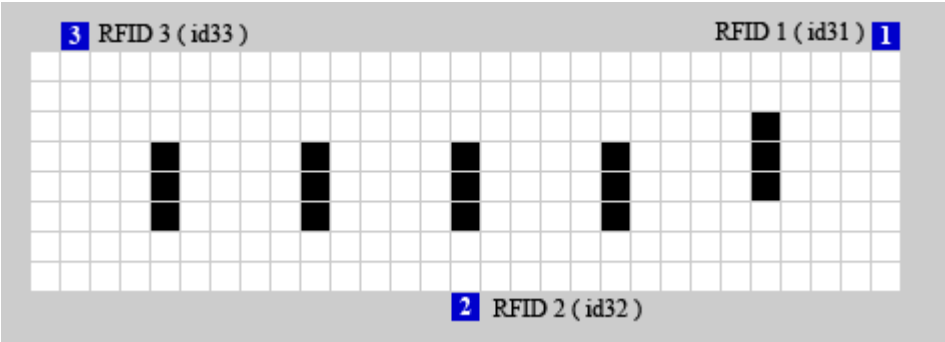


Figure 4.27: Places of the 3 RFID beacons in experiment area

In Trilateration technique, it is possible to calculate the unknown location by using other 3 known ones (Figure 4.28).

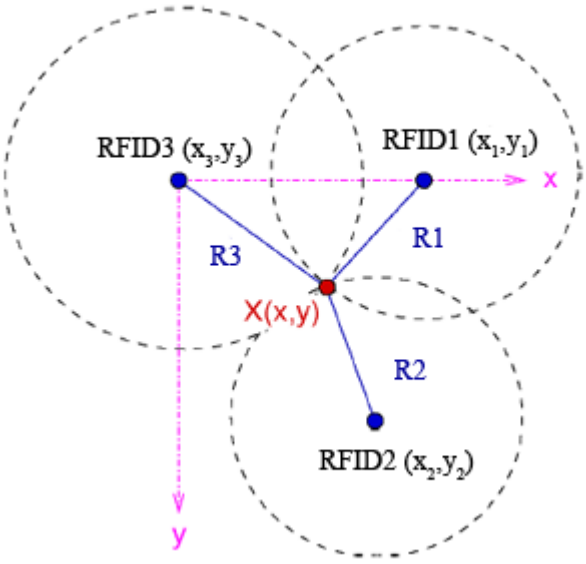


Figure 4.28: Trilateration

Formulas used to calculate the positions of the unknown point are given in Figure 4.29.

$$x^2 + y^2 = R3^2$$

$$(x - x_1)^2 + y^2 = R1^2$$

$$(x - x_2)^2 + (y - y_2)^2 = R2^2$$

$$x = \frac{x_1^2 + R3^2 - R1^2}{2x_1}$$

$$y = \frac{x_2^2 + y_2^2 + R3^2 - R2^2 - 2xx_2}{2y_2}$$

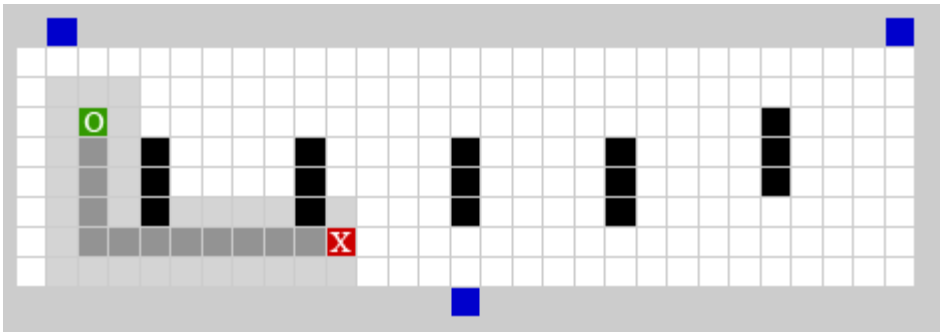
Figure 4.29: Trilateration Formulas

Trilateration is a method of determining the relative positions of objects using the geometry of triangles. Unlike triangulation, which uses angle measurements (together with at least one known distance) to calculate the subject's location, trilateration uses the known locations of two or more reference points, and the measured distance between the subject and each reference point. To accurately and uniquely determine the relative location of a point on a 2D plane using Trilateration alone, generally at least 3 reference points are needed.

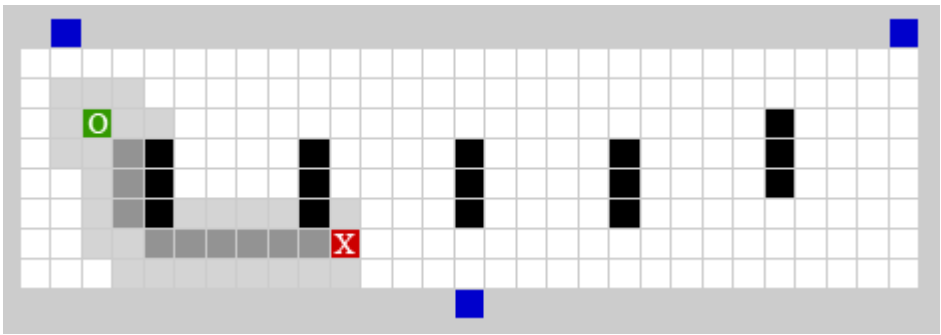
4.4 Comparison of Heuristics

We took several measurements in simulation (offline) mode, and the results can be seen in Figure 4.30, Figure 4.31 and Figure 4.32.

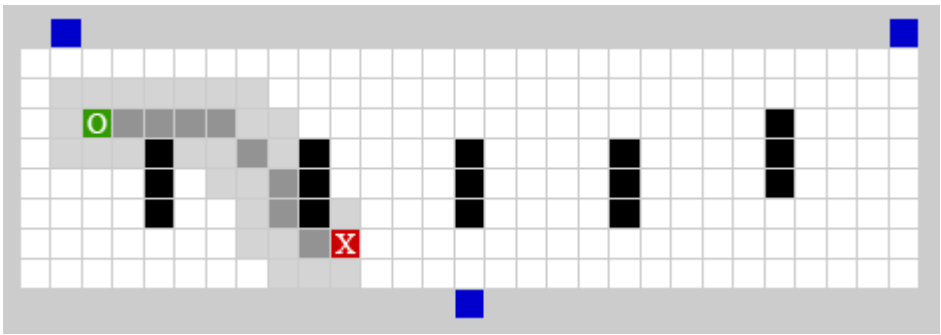
Sample 1:



(a) Manhattan, 10.744 ms, 12 cells to go



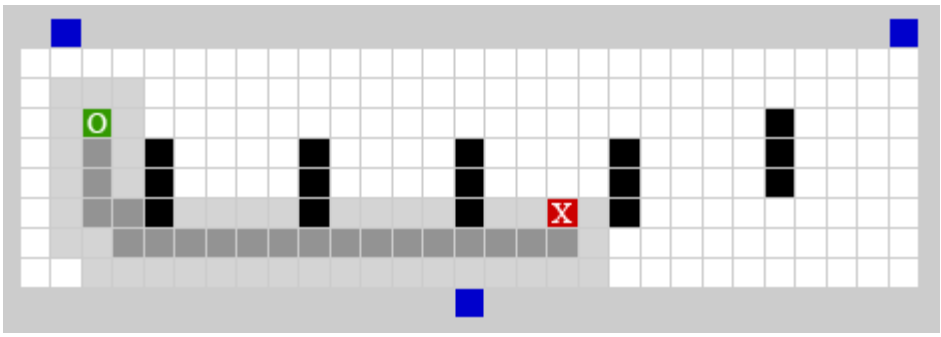
(b) Euclidean, 5.640 ms, 10 cells to go



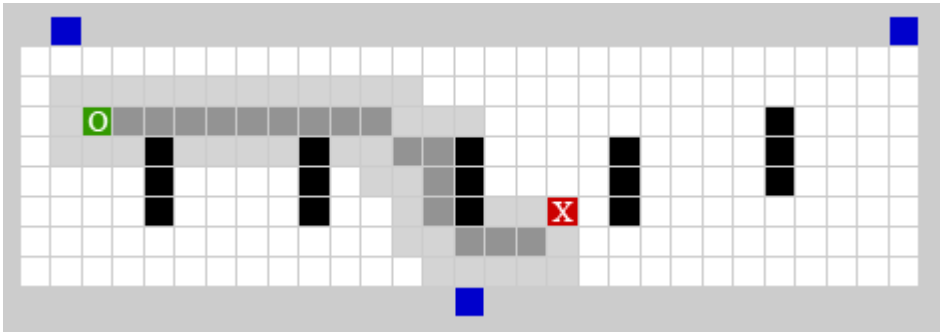
(c) Diagonal, 7.659 ms, 9 cells to go

Figure 4.30: Comparison of different Heuristics and Calculation Times – 1

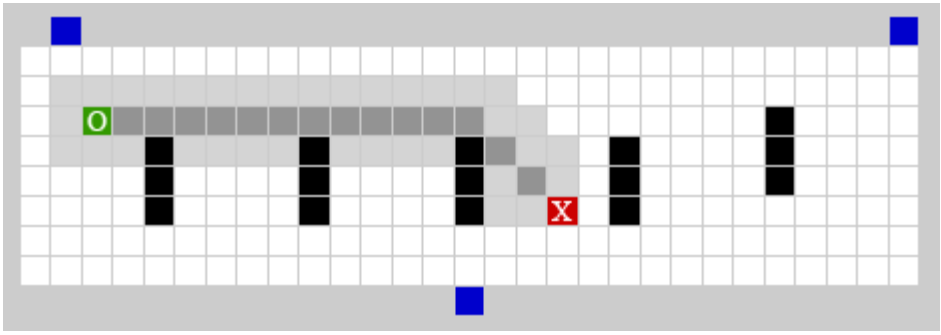
Sample 2:



(a) Manhattan, 6.176 ms, 20 cells to go



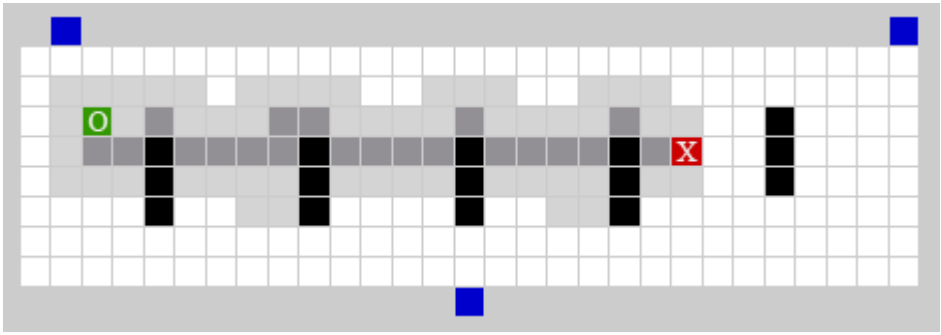
(b) Euclidean, 5.160 ms, 17 cells to go



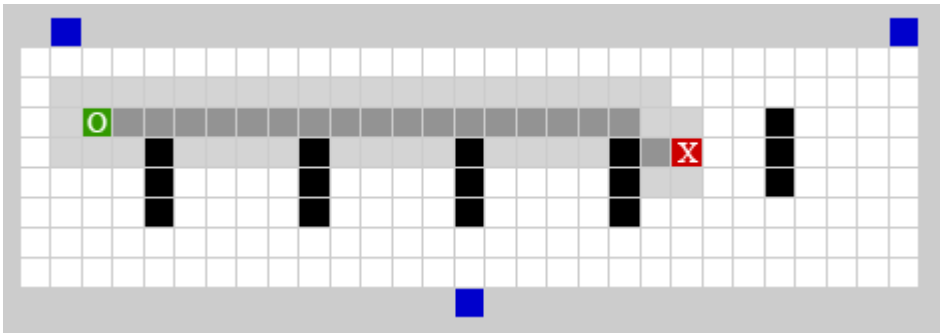
(c) Diagonal, 4.762 ms, 15 cells to go

Figure 4.31: Comparison of different Heuristics and Calculation Times – 2

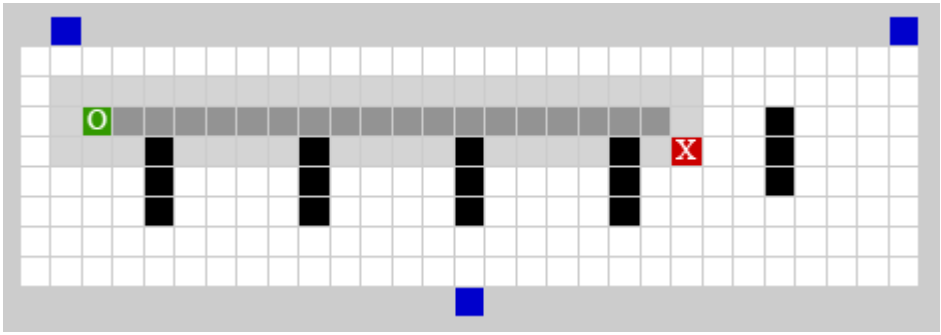
Sample 3:



(a) Manhattan, 12.668 ms, 21 cells to go



(b) Euclidean, 6.074 ms, 19 cells to go



c) Diagonal, 5.888 ms, 19 cells to go

Figure 4.32: Comparison of different Heuristics and Calculation Times – 3

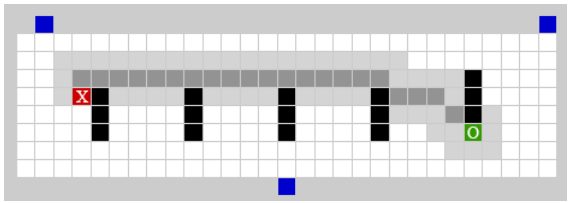
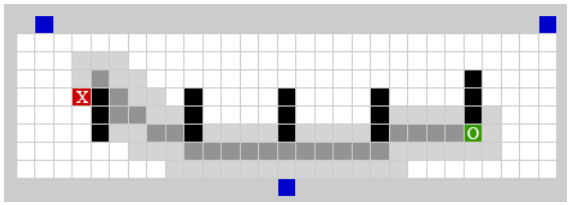
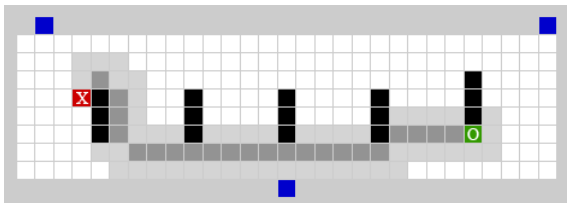
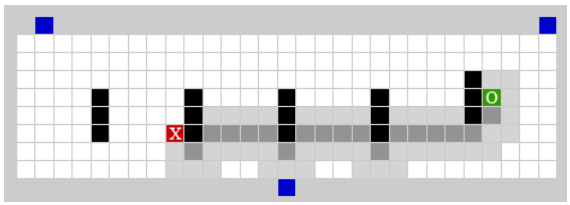
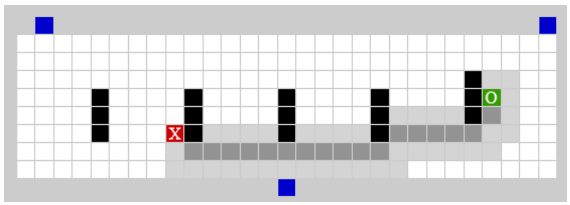
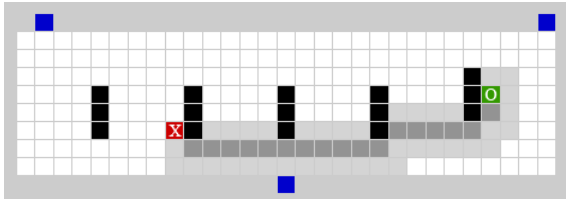
Sample 4		
	Manhattan	19.935 ms
	Euclidean	17.480 ms
	Diagonal	10.188 ms
Sample 5		
	Manhattan	5.227ms
	Euclidean	6.064 ms
	Diagonal	5.664 ms

Table 4.3: Comparison of different Heuristics and Calculation Times – 4

As we can see from these results, Diagonal type Cost Calculation Method gives the best result for finding the path in terms of time passed and the path length which we will guide the user to walk on. By shortening the path, we can prevent to make unnecessary calculations and also we can forward the user to the destination faster. In Table 4.4, sample data is summarized.

Sample Number	**Manhattan**		**Euclidean**		**Diagonal**	
	Time (ms)	***Path Length***	***Time***	***Path Length***	***Time***	***Path Length***
1	10.744	12 cells	5.64	10 cells	7.659	9 cells
2	6.176	20 cells	5.16	17 cells	4.762	15 cells
3	12.668	21 cells	6.074	19 cells	5.888	19 cells
4	19.935	22 cells	17.48	22 cells	10.188	23 cells
5	5.227	18 cells	6.064	18 cells	5.664	18 cells
6	12.326	26 cells	10.32	23 cells	7.233	24 cells
7	2.671	9 cells	2.016	6 cells	1.795	7 cells
8	2.799	13 cells	2.6	13 cells	2.153	11 cells
9	1.242	8 cells	1.232	7 cells	1.602	8 cells
10	2.557	17 cells	3.169	16 cells	2.468	16 cells
11	1.358	12 cells	1.322	11 cells	1.281	11 cells
12	4.617	12 cells	0.993	8 cells	0.837	6 cells
13	4.415	26 cells	3.028	23 cells	3.897	21 cells
14	2.816	18 cells	1.679	14 cells	1.447	14 cells
15	1.512	10 cells	0.777	6 cells	0.95	7 cells
16	1.874	16 cells	1.871	16 cells	1.713	15 cells
17	0.903	9 cells	0.951	9 cells	0.833	8 cells
18	0.547	5 cells	0.59	4 cells	0.633	5 cells
19	2.185	17 cells	2.156	16 cells	1.841	16 cells
20	2.373	21 cells	3.204	22 cells	3.849	20 cells

Table 4.4 Sample Simulation Data

We can see from these results that, in overall, Diagonal method is 17.8% faster than Manhattan method and 3% faster than Euclidean method.

Diagonal method also produces smoother paths; this can be very useful since we try to guide a visually impaired person. We give less coordinates to the user by this method, which reduces the probability of being on the wrong path/cell. Since the user walks a shorter path, s/he can arrive to the desired destination quicker.

Since the errors in RFID measurements are greater than our small size on the map, size of a cell 60cm X 60cm, a need for a checking method is raised. So we have implemented the Bayesian type prediction method to our path finding system. These methods are first implemented in Electronics Department by using a Technical Computing program, MATLAB®, and then we converted it to the Java Language. When implementing this method, we met with a problem. This problem comes from the neighboring cells having the same RSS values. We solved this problem by distributing the RSS values to each individual cell. However, since the measurements from the neighboring cells do not differ much, it is not possible to completely identify the cells by using RFID technology. It may differ if we collect much more data for the same cell, and then analyze these data. However, since the user moves with the directions and the speed of a moving person (even s/he is vision disabled) is high to acquire much more data, this idea may not be applicable to our system.

Our last studies show that the path found from the Bayes does not differ much from the path found with A* Algorithm, it is in the acceptable limits.

We also implemented the communication functions with the RFID transceiver by using RS232 port of the computer (or using a USB port with a RS232 converter). We get the RFID RSS values from the Serial Port and use them for finding the location of the HHNU (and its user). However the formulas we used for calculating the distance values by using RSS values should be revised, because they could not give the correct result. This error also affects the Trilateration method we used for finding the location of the user.

CHAPTER 5

Conclusion and Suggestions

The main aim of this study was to develop a utility that can communicate with the outer systems (subsystems of the PUSULAL), and implement a path finding algorithm to the developed utility. Our major purpose was to design the administrator interface of a navigation aid for the visually impaired people when they go shopping.

In order to realize this aim, first we made a literature survey in order to see what has been done before. We saw that there is quite a lot of study to make the life of visually impaired people easier. Some researchers studied on developing robotic aids, which consists of some moving devices. These devices have all of the location and obstacle sensors, guiding systems and processing units on itself. Because of all of these features, these systems are big in size, at least the size of a small shopping basket.

There are also some other studies including smaller units that can be carried on the user's body. They have big antennas to measure and find the location of the user, generally GPS which enables outdoor usage.

Some of the works encountered in the literature use the RFID technology; they are designed especially to follow a road to reach a known destination. These can be used to guide the user between two outdoor places, for example from dormitory to library or from house to the shopping center.

There also exist some indoor studies which use some other technology, like Infrared, Laser or Ultrasound transceivers. These systems search the environment, make the map of their environments, find the obstacles on the way and guide the user according to the knowledge they collect.

We have seen that all of these systems have useful properties separately, but what happens in the case, where the user is in the shopping center or market and wants to handle the shopping by him/her self. We focused on this problem and tried to find a solution by using RFID technology in a closed well-known environment.

Since this study is a part of PUSULAL project, we worked with the Electrics and Electronics Department to make this study become real. Our job includes the implementation of a path finding algorithm and communicating with the devices that is developed in the Electronics Department and the instruments that are obtained from the market.

In order to develop a utility that can work on different operating systems and platforms such as Windows, Linux and UNIX, we have selected the Java Language. It is powerful, Object Oriented, independent of the OS and widely used in the world. We developed our entire interface in Netbeans IDE by using the standard and extended libraries of Java Language.

Next we decided on the Path Finding Algorithm that we will use in our system. After the literature search we decided to use A* (A Star) Algorithm, since it is the best choice in environments like ours. We implemented this algorithm with three different Cost Calculation Methods to our system, which gives us the option of comparing their differences.

This study is the main skeleton or the nervous system of the PUSULAL project: it makes the whole communication processes with the subsystems. At the time being, since the other subsystems of the PUSULAL are not completely finished, we could not be able to try our system on a real world environment. We believe that this study can hold a light to the future studies and the future developments with this project. After implementing and connecting the subsystems completely, it can be clear to find the real time results.

5.1 Future Study

As a future study, multiple destinations way finding or path planning can be studied. This work may consist of providing an option to the user for entering multiple targets and then calculating a shopping route having the most shortest walk path.

It is also possible to implement some other Path Finding Algorithms, such as D*, when normal/base system is studied in the experiment area and some other moving obstacles or other users enter the experiment area.

REFERENCES

[1] Carsten J., *Global Path Planning on board the Mars Exploration Rovers*, IEEE Aerospace Conference, 2007

[2] Obara T., Yamamoto K., Ura T., Maeda H., Yamato H., *Development of an Autonomous Underwater Vehicle R1 with a Closed Cycle Diesel Engine*, Proc. ISOPE 94, Vol.2, Apr. 1994, pp.351-357

[3] David M. Bourg, Seeman G., *AI for Game Developers,* O'Reilly, Chapter 7, June 2004.

[4] M. Shafie Abd. Latiff, and R. Hassan, *An Efficient Virtual Tour- A Merging of Path Planning and Optimization*, Work with Computing Systems Conference, Kuala Lumpur, 2004.

[5] T.Fröhlich and D. Kullmann, *Autonomous and Robust Navigation for Simulated Humanoid Characters in Virtual Environments*, Proceedings of the First International Symposium on Cyber Worlds (CW'02)

[6] Adam A. Razavian, Sun J., *Cognitive Based Adaptive Path Planning Algorithm for Autonomous Robotic Vehicles*,, SoutheastCon 2005 Proceedings, 8-10 April 2005.

[7] Koenig S. and Likhachev M., *Improved Fast Replanning for Robot Navigation in Unknown Terrain*, Proceedings of the 2002 IEEE, International Conference on Robotics & Automation,Washington DC, May 2002.

[8] Yahja A., Singh S., Stentz A., *An efficient on-line path planner for outdoor mobile robots*, Robotics and Autonomous Systems 32 (2000), Elsevier Science.

[9] Jacob R., Marathe M. and Nagel K., *A Computational Study of Routing*

Algorithms for Realistic Transportation Networks, ACM Journal of Experimental Algorithms Vol 4 No 6 (1999)

[10] Kulyukin V., Gharpure C., Nicholson J. and Pavithran S., *RFID in Robot-Assisted Indoor Navigation for the Visually Impaired*, Proceedings of 2004 lEEE/RSJ International Conference on Intelligent Robots and Systems September 28-October 2, 2004, Sendai, Japan

[11] Amemiya T., Yamashita J., Hirota K. and Hirose M., *Virtual leading blocks for the deaf-visually impaired: a real-time way-finder by verbal-nonverbal hybrid interface and high-density RFID tag space*, Virtual Reality, 2004. Proceedings. IEEE, 27-31 March 2004.

[12] Ran L., Helal S. and Moore S., *Drishti: An Integrated Indoor/Outdoor visually impaired Navigation System and Service*, In Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (PerCom.04), pp. 23-30, 2004.

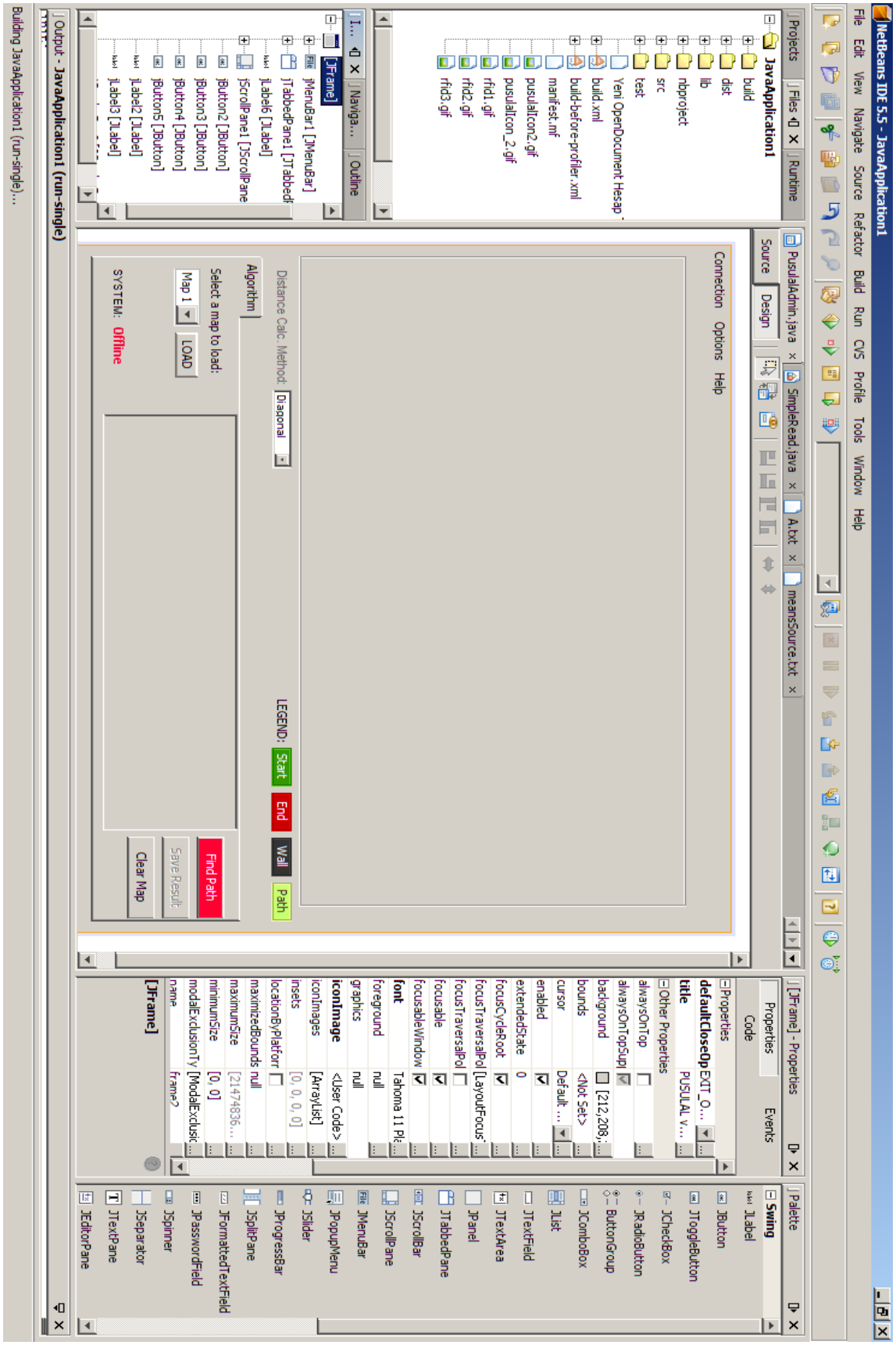
[13] E. W. Dijkstra, *A note on two problems in connexion with graphs,* Numerische Mathematik. 1 (1959), S. 269–271

[14] Patel A., *Amit's Game Programming Site*, last access: 15th of August 2007 http://theory.stanford.edu/~amitp/GameProgramming/index.htm,

[15] Hart P. E., Nilsson N. J., Raphael B., *A Formal Basis for the Heuristic Determination of Minimum Cost Paths,* IEEE Transactions on Systems Science and Cybernetics SSC4 (2): pp. 100–107, 1968.

[16] Dinçer Z., Öktem R., Aydın E., *Asş'ne Dayalı Konum Kestirimi İçin Bir Bayes Karar Modellemesi*, IEEE 15th Signal Processing and Communication Applications Conference SIU 2007.

APPENDIX A: NETBEANS IDE SCREENSHOT

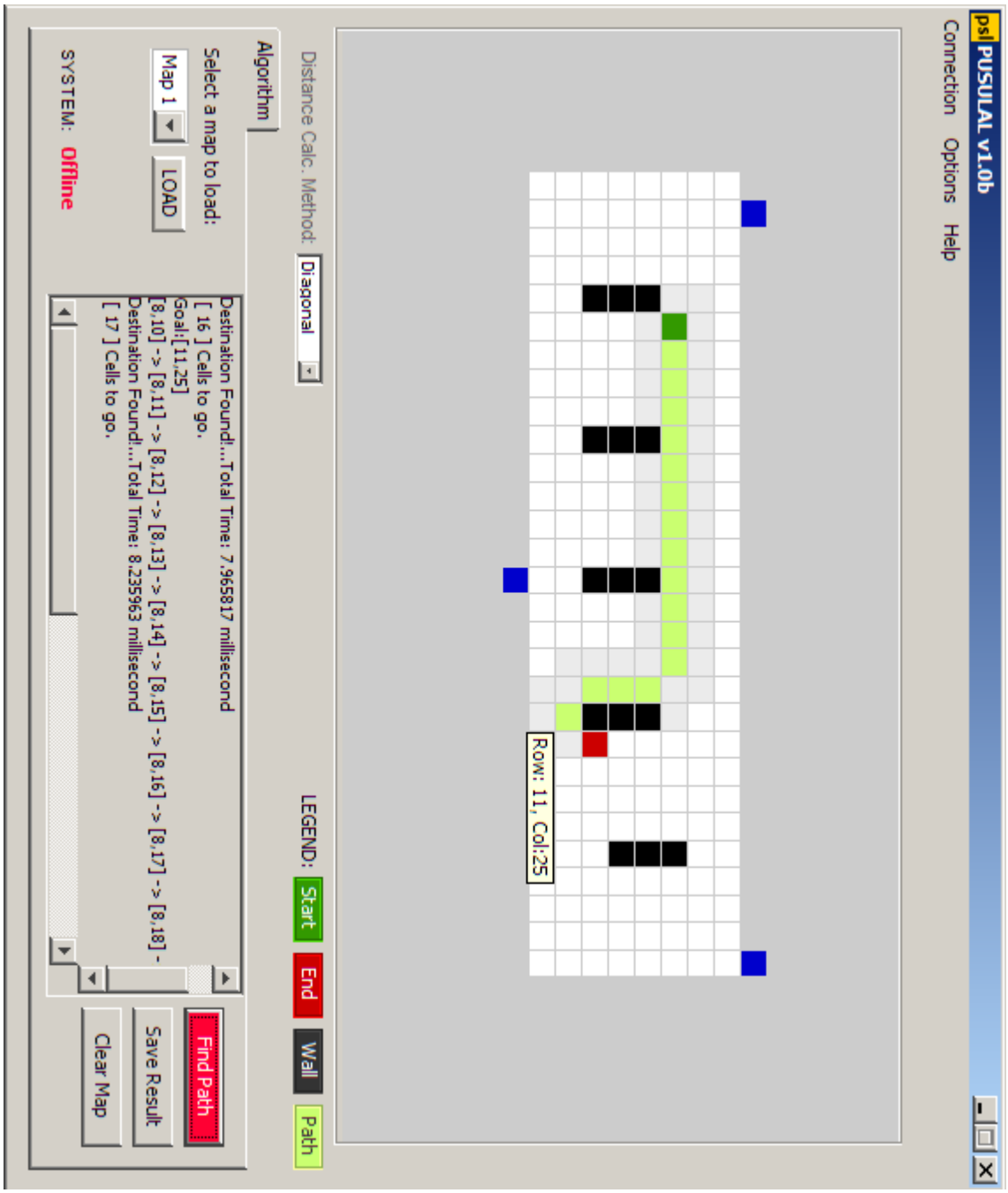


APPENDIX B: JAVA CODE FOR A* SEARCH ALGORITHM

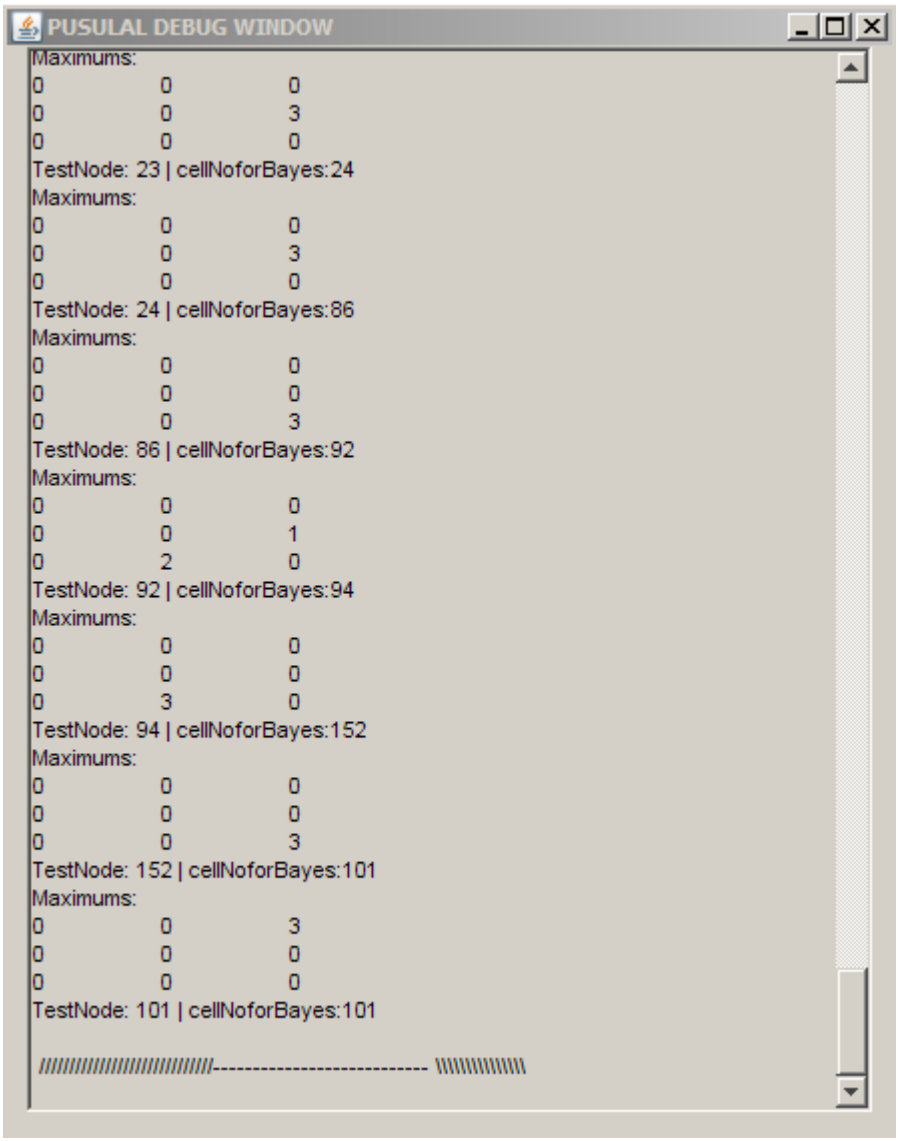
```
private void AStarSearch(mynode Start, mynode Goal){
      // gecen zamani bulmak icin timer basliyor
      long startTime=System.nanoTime();

      // listeler temizleniyor.
      OpenList.clear();
      ClosedList.clear();

      double g = 0;
      double h = Start.getCostTo(Goal,costModel);

      Start.setScore(g+h);
      OpenList.add(Start);

      mynode bestNode;
      mynode newNode = null;

      while(OpenList.size()!=0){
         bestNode = OpenList.poll();

         // hangi node lara gidilecek onu gosteriyor
         jTextArea1.append("["+Integer.toString(bestNode.getRow())+","
                           +Integer.toString(bestNode.getCol())+ "]" +" -> ");

         // bulunan hucre hedef ile ayni ise...
         if (bestNode.getCostTo(Goal,costModel)==0){
            long elapsedTime = System.nanoTime()-startTime;
            jTextArea1.append("\nDestination Found!...");
            jTextArea1.append("Total Time: "+elapsedTime/1000000.0
                              +" millisecond\n");
           mynode dummy=new mynode(bestNode.getRow(), bestNode.getCol(),
                                   bestNode.getParent());
            mynode dmm=dummy;
            int totalCells=0;
            while (dummy!=Start){
                  if ((dummy.getRow()!=Goal.getRow()) ||
                      (dummy.getCol()!= Goal.getCol())){
                      jTable1.setValueAt(new Color(202,255,112), dummy.getRow(),
                                         dummy.getCol());
                  }
                  totalCells=totalCells+1;
                  dummy=dummy.getParent();
                  dmm=dummy;

            }
            jTextArea1.append(" [ "+totalCells+ " ] Cells to go.\n");
            break;
         }

         // baslangic=hedef degil ise etraftaki hucreleri arastir...
         else {
            int nodeX, nodeY;
            mynode parent;
            ClosedList.add(bestNode);
```

```
            for (int i=0; i < 8; i++) {
               if (i==1){
                  nodeX = bestNode.getRow()-1; nodeY = bestNode.getCol()-1;
               } else if (i==2){
                  nodeX = bestNode.getRow();   nodeY = bestNode.getCol()-1;
               } else if (i==3){
                  nodeX = bestNode.getRow()+1; nodeY = bestNode.getCol()-1;
               } else if (i==4){
                  nodeX = bestNode.getRow()+1; nodeY = bestNode.getCol();
               } else if (i==5){
                  nodeX = bestNode.getRow()+1; nodeY = bestNode.getCol()+1;
               } else if (i==6){
                  nodeX = bestNode.getRow();   nodeY = bestNode.getCol()+1;
               } else if (i==7){
                  nodeX = bestNode.getRow()-1; nodeY = bestNode.getCol()+1;
               } else {
                  nodeX = bestNode.getRow()-1; nodeY = bestNode.getCol();
               }
               parent = bestNode;
               if (((nodeX>=0) && (nodeX<=20)) && ((nodeY>=0) && (nodeX<=39)))
                newNode = new mynode(nodeX,nodeY,parent);

               double newg= newNode.getCostFrom(bestNode,costModel);
               double newh = newNode.getCostTo(Goal,costModel);
               newNode.setCostFrom(newg);
               newNode.setCostTo(newh);
               newNode.setScore(newg+newh);

 if ((newNode.getRow()!=Start.getRow()) || (newNode.getCol()!=Start.getCol()))
  if ((newNode.getRow()!=Goal.getRow()) || (newNode.getCol()!=Goal.getCol()))
         if (NodeMap[nodeX][nodeY]!=3)
              jTable1.setValueAt(new Color(235,235,235), newNode.getRow(),
                                        newNode.getCol());
       if ((OpenList.contains(newNode)==true) &&
      (newNode.getCostFrom(Start, costModel) >=bestNode.getCostFrom(Start, costModel))) {
                 jTextArea1.append("NODE IN OPEN LIST\n");
                 continue;
               }
 else if ((ClosedList.contains(newNode)==true) &&
  (newNode.getCostFrom(Start, costModel) >= bestNode.getCostFrom(Start, costModel))) {

                 jTextArea1.append("NODE IN CLOSED LIST\n");
                 continue;
               } else {
                 if (NodeMap[nodeX][nodeY]!=3)  {
                    OpenList.add(newNode);
                 }
               }
            }//end of for

            if (probability==true)
               checkPath(bestNode);
            ClosedList.add(bestNode);

         }//end of else
      }
   }
```

APPENDIX C: PUSULAL Admin Screenshot



APPENDIX D: PUSULAL Admin Debug Window Screenshot



APPENDIX E: RSS Contents of the Cell A

(First column is the measurement ID, other three columns are the RSS values coming from 3 different RFID beacons.

11	52	38	42
12	51	43	38
13	49	43	40
14	49	40	41
21	49	43	43
22	52	45	42
23	47	40	41
24	51	42	39
31	41	47	40
32	46	46	39
33	47	48	38
34	46	39	41
41	49	48	44
42	54	49	43
43	46	45	43
44	47	45	43
51	65	45	42
52	48	46	38
53	52	40	43
54	53	35	42
61	65	46	43
62	45	43	40
63	49	41	41
64	49	45	39
71	52	49	43
72	52	45	43
73	53	47	44
74	51	44	43
81	52	45	41
82	51	46	41
83	54	41	39
84	52	45	43
91	51	43	40
92	65	43	37
93	52	40	29
94	50	43	43
101	52	45	45
102	47	46	43
103	50	43	43
104	49	45	41
111	53	45	44
112	45	44	36
113	43	42	40
114	49	40	43
121	53	48	44
122	53	45	41
123	65	45	44
124	51	48	42
131	64	45	41
132	55	42	41
133	50	43	42
134	52	45	37

APPENDIX F: RSS Contents of the Cells that were parts of Cell A

1.txt

11	52	38	42
12	51	43	38
13	49	43	40
14	49	40	41
21	49	43	43
22	52	45	42
23	47	40	41
24	51	42	39
31	41	47	40
32	46	46	39
33	47	48	38
34	46	39	41
41	49	48	44
42	54	49	43
43	46	45	43
44	47	45	43
51	65	45	42
52	48	46	38
53	52	40	43
54	53	35	42

2.txt

14	49	40	41
21	49	43	43
22	52	45	42
23	47	40	41
24	51	42	39
31	41	47	40
32	46	46	39
33	47	48	38
34	46	39	41
41	49	48	44
42	54	49	43
43	46	45	43
44	47	45	43
51	65	45	42
52	48	46	38
53	52	40	43
54	53	35	42
61	65	46	43
62	45	43	40
63	49	41	41

3.txt

24	51	42	39
31	41	47	40
32	46	46	39
33	47	48	38
34	46	39	41
41	49	48	44
42	54	49	43
43	46	45	43
44	47	45	43
51	65	45	42
52	48	46	38
53	52	40	43
54	53	35	42
61	65	46	43
62	45	43	40
63	49	41	41
64	49	45	39
71	52	49	43
72	52	45	43
73	53	47	44

4.txt

82	51	46	41
83	54	41	39
84	52	45	43
91	51	43	40
92	65	43	37
93	52	40	29
94	50	43	43
101	52	45	45
102	47	46	43
103	50	43	43
104	49	45	41
111	53	45	44
112	45	44	36
113	43	42	40
114	49	40	43
121	53	48	44
122	53	45	41
123	65	45	44
131	64	45	41
132	55	42	41