Indoor Routing in Three Dimensional Spaces (Case Study: Telkom University Bandung)

Final Project

Tiara Annisa Dionti
1103134405

Bachelor of Informatics Engineering

School of Computing

Telkom University

Bandung

2017

Approval Sheet

Indoor Routing in Three Dimensional Spaces
(Case Study: Telkom University Bandung)

Tiara Annisa Dionti
NIM: 1103134405

This final project has been accepted and approved to fulfill most of the requirements to obtain a Bachelor's Degree of Informatics Engineering
School of Computing
Telkom University

Bandung, January 3rd 2017
Signed by,

Supervisor 1	Supervisor 2
Kiki Maulana Adhinugraha, Ph.D.	Sultan Mofareh Alamri, PhD.
NIP: 06800352-1	

Approved by,
Head of Undergraduate

Arif Bijaksana, Ph.d
NIP: 95650581-1

Abstract

Spatial data is the data that stores geographic data types. This data is often used on systems that use data related to the territory of a region, such as the routing system or navigation system. The routing system itself has been implemented on the outdoor routing, and over the times began to be developed in the direction of the indoor routing. There are significant differences that make routing more indoor than outdoor complex routing, which is on the outdoor routing only implement routing in two dimensional spaces, while at the indoor routing allows the routing of the three dimensional spaces that represent high rise building. In the case of indoor routing, using the method of Three Dimensional Spaces will identify an object accurately by storing spatial data are represented to form undirected graph or graph is not directed to the attributes of three-dimensional data where x and y are the coordinates of a point, and z represents level height of the point. Indoor use the shortest path routing algorithm can be implemented after the three dimensional spaces structure was built in order to provide output that can be taken the shortest route between two points. This final project aims to implement routing indoor systems using the method of three dimensional spaces on School of Computing, Telkom University's spatial data to be able to determine the shortest route in general and close shortest route wich means the shortest route that will not pass through the open space.

Keywords: Spatial, Indoor Routing, Three Dimensional Spaces, Graph, Shortest Path Algorithm

Declaration Sheet

I hereby certify that the final project entitled "Indoor Routing in Three Dimensional Spaces (Case Study: Telkom University Bandung)" and all its contents is really my own work and I do not plagiarism, quoting a way that does not match the ethics of science, as well as plagiarism. On this statement I am ready to pay the penalty or sanction, if indeed a violation found in my work.

Bandung, January 3rd 2017
Signed,

Tiara Annisa Dionti

Acknowledgements

Preface

Assalamu'alaykum Wr. Wb.

All praise to Allah for over abundance of His grace so this project about Indoor Routing in Three Dimensional Spaces (Case Study: Telkom University Bandung) can be resolved. This final project was made as one of requirements to obtain a Bachelor's Degree of Computer Science in Telkom University. It is undeniable that this final project is certainly still not perfect enough. However, the author strive to provide the best in this final project.

Hopefully this final project will be useful to the readers. Criticism and suggestions from readers of this book will be gladly accepted to make this research can be developed even better in the future.

Wassalamu'alaykum Wr. Wb.

Bandung, January 3rd 2017
Writer

Contents

List of Figures

List of Tables

Chapter I

Introduction

1.1 Overview

In recent years, a navigation system or outdoor routing systems like Google Maps is become very beneficial, especially for people who travels alot without knowing the direction to go towards their destination place [4]. Later, this navigation system is also implemented in a smaller area, like mapping on indoor spaces. With the indoor mapping system, someone will be facilitated in finding the target location[6]. Indoor routing application is pretty much grown abroad for certain places such as malls, airports, offices, etc [1]. However in Indonesia, the majority still use manual mapping system by displaying a room map plan in the building.

There are significant differences that make indoor routing more complex than outdoor routing, which is on the outdoor routing generally implemented in two dimensional space, while at the indoor routing allows the routing of the three dimensional space that represent multi-storey building [6]. This is a challenge in constructing indoor routing system, how to represent indoor spaces in a building? A building may have a number of rooms and the number of corridors. From every room allows for a variety of door that connect the room with another spaces. And the buildings can be a multi-storey building that has stairs, lifts, or elevators to move from one level to another level. The entire space must have identified labels as well as connectivity between the spaces. Three dimentional spaces can be a solution to build the system's indoor routing. This method will identify an object accurately by storing geographic data are represented to form undirected graph or graph is not directed to the attributes of the data of three-dimensional x, y, and z, where x and y are the coordinates of a point, and z represents the height level point the [2].

In this thesis, indoor routing by using a three-dimensional representation spaces construction will be implemented for the case study the buildings of the Faculty of Informatics of Telkom University. This system is expected to provide optimal service that can be taken to achieve the intended room.

1.2 Major Challange

Based on the background described above, problems can be formulated as follows:

1. How does the representation of three dimensional spaces can be implemented to build a data structure of the buildings of Faculty of Informatics, Telkom Universty?
2. How to implement indoor routing in three dimensional spaces?
3. How does the performance of the indoor routing system using three dimensional spaces?

1.3 Objective

The objectives to be achieved in this final project are:

1. Modeling the 3D coordinates of multi-floored building.
2. Plotting the coordinates the derrived from the earth coordinates.
3. Implementing the road network algorithm in 3D model.
4. Analyzing the performance of road Network algorithm in 3D model.

1.4 Scope

The scopes of this final project are:

1. Graph representation of data that used in this final project is the undirected graph.
2. The route chosen is the shortest route with two options, the shortest route in general and the enclosed one (roofed overall). If the enclosed route is not found, it raised only the shortest route.
3. The data set used is the Faculty of Computer Science, Telkom University's spatial data (only along D, E, and F buildings).

1.5 Hypothesis

Three dimensional spaces is a method for representing spatial data. The routing system method by applying two-dimensional space should also apply to three dimensional space. The system will be able to receive two inputs, the first input is the location of starting point and the second input is the destination point. The system would then output the shortest route from those points. There are two kind of shortest route that will be outputted, the shortest route in general and the enclosed one. If the enclosed route is not found, it raised only the shortest route.

1.6 Methodology

The settlement method will be used to complete this final projet are:

1. Literature Study

   Studying the literature that can be used as a reference regarding Indoor Routing with three dimensional spaces.

2. Data Collecting and Analysis

   Authors collected the dataset that will be used to implement the three dimentional spaces for Indoor Routing. At this stage, there will be labeling on every point included in the dataset.

3. System Model Development

   This stage includes needs analysis, development analysis, and modeling indoor routing system with arithmetic models.

4. System Development

   This stage includes the development of software in accordance with the design of the previous stage.

5. System Testing

   Perform testing of the system in terms of accuracy and performance of the system.

6. Analysis and Conclusions

   Analyzing the results of the accuracy of three dimensional spaces construction.

1.7 Summary

This chapter is about the fundamental of what this final project about. From the problems stated above, this final project offer a solution. Again, the main problem is how to build the suitable data structure for indoor routing system. This final project offers to use three dimensional data structure as a solution.

Chapter II

Literature Review

2.1 Spatial Database

Spatial database is a database designed to store and process data with spatial data types such as point, line, or region contained in Geographic Information System (GIS). Spatial data can be utilized on a two dimensional appearance as the appearance of the earth's surface, or the appearance of three dimensions, such as modeling the human brain, the protein molecule chains, etc. After the systems of relational database developed, the system of other databases become developed then, one of it is the spatial database. Characteristics of spatial database needed is a system that is able to accommodate simple geometric objects data with a huge amount to accommodate as 100,000 polygons.

Spatial database system is gaining popularity in recent years, especially at a "Symposium on Large Spatial Databases (SSD)" held biennially since 1898 associated with the database that stores the object in space as a supporter of space images [9].

2.1.1 Data Representation

As mentioned previously, spatial data can be represented by a point, line, or region. For example, a city can be modeled as a set of points that describe a large geographical area. A line usually used to represent the connections in space such as roads, rivers, telephone cables, electricity, etc. An object representing the area or region that has a limit in two-dimensional space, such as the state, lakes, etc. Figure 2.1 is a basic representation of an image on a spatial database, which is the point, line, and area.

Figure 2.1: Three basic spatial representations: points, lines, areas

Spatial objects relate to each other can be described by the form of partitions or networks as shown in Figure 2.2 below.

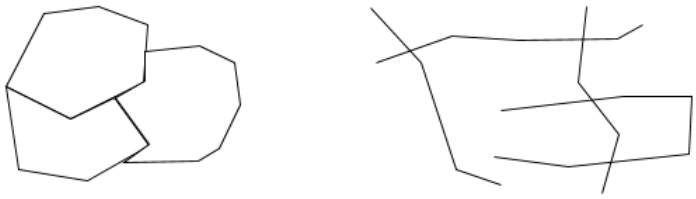

Figure 2.2: Partitions and networks

A partition can be described by a group of separate region. The liaison between the regions is the regions' equally boundary. For example, the partition can be used to represent a thematic map. A network can be illustrated by the graph that are connected in a field where each object point considered as nodes and lines as the geometric shapes of the edges. This network forms can be used to represent roads, rivers, public transportation, or electricity cable lines [5].

2.2 Routing System

Routing System is a system that provides driving directions or routes that can be taken by a moving object in order to reach a location of the destination. This system is often referred to the navigation system. Modern navigation systems now have integrated between the position of the object, sensors, computing, and communications between the hardware and software to support the facilities in humans, vehicles, and other moving objects. In addition, modern navigation systems also consider the geographical location coordinates distance accuracy, speed, and altitude of the moving object. The navigation system is widely used in outdoor data. But along with the times, this system is also applied to the indoor data.

2.2.1 Outdoor Routing

Outdoor routing technology is now highly developed. Figure 2.3 describes the flow of information on outdoor navigation systems.

From Figure 2.3, the user's position is determined by: (a) obtaining position data through geo-positioning sensor and (b) applying the map matching algorithm using position data obtained. These steps are general steps to improve the accuracy, availability, and reliability of outdoor navigation system by using more than one geo-positioning sensor in which each of the position data can be filtered using a Kalman filter to find the best position estimation. Furthermore, the position data that has been filtered out will be input to the map matching algorithm that uses a database map for traveling the area,

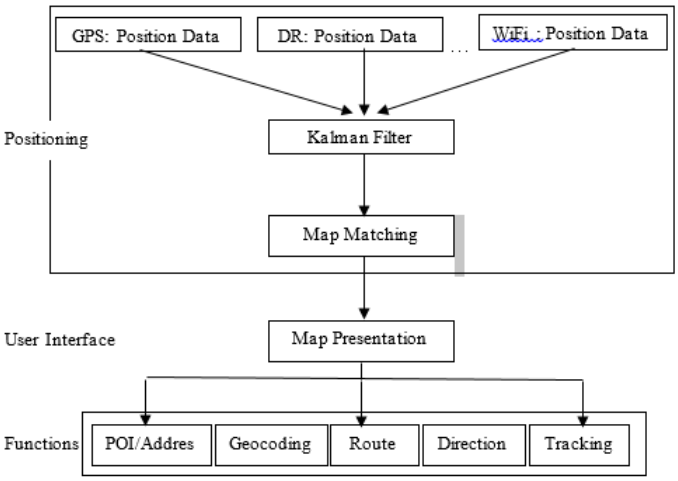

Figure 2.3: The information flow on outdoor navigation systems

which contains spatial and non-spatial to find: (a) roads / pavements where users are located and (b) the location right of user in the segment.

When the user's location is known, the locations will be marked in the map and displayed to the user. At this stage, the system is currently tracking, and users have the option to search for POIs or a request for an optimal route between the pair of addresses. The system uses a search using the shortest or fastest route criteria[7]. Figure 2.4 is an example of outdoor routing system implementation.

Figure 2.4: Examples of outdoor routing implementation

2.2.2 Indoor Routing

Indoor routing is a routing system implementation in the room or building. By using spatial data, each space can be accurately identified. There are

various kinds of elements in indoor spaces such as rooms, doors, corridors, floors, stairs, elevators, and the road connection between buildings. Indoor spaces are represented by undirected graph with nodes and edges. The concept of modeling the indoor spaces can be summarized in Table 2.1.

Table 2.1: The concept of indoor spaces modeling

Domain Concept	**Modeling Concept**
Room	A cell
Door	An edge
Corridor	One or more cells with one or more edges
Stair	One or more cells with one or more edges
Elevator	One cell with several edges
Pathway	One or more cells with several edges

As an illustration, Figure 2.5 is a graph modeling the indoor spaces delineated by one level.

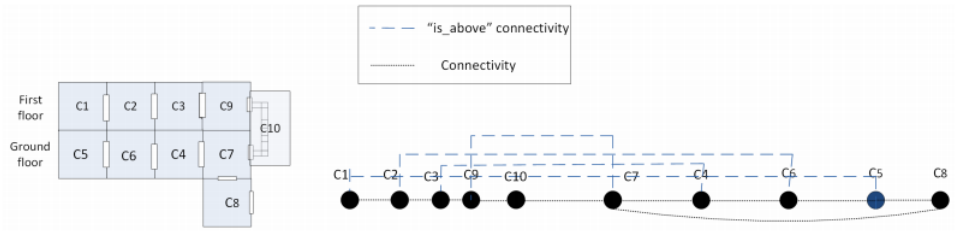

Figure 2.5: Modeling indoor spaces with a graph of the level

From the illustration, it can be seen that C10, C9, C8, C7, C4, C6, C5, C3, C2 and C1 are connected where C10 is the ladder that connects the ground floor and the first floor. Given this concept, it can be concluded that:

1. An indoor space is a connected undirected graph (Cells, Edges), where Cells = C1, C2, ..., Cn is a set of cells, and Edges = E1, E2, . . . , Em is set of edges, each of which is a set of two cells, that is, each edge connects two distinct cells.

2. (Adjacent Nodes) Let G = (N,C) be a graph, and two Nodes n1,n2 $\in$ N of G are adjacent if: $\exists$ n1,n2 are connected.

3. (Multidimensional Connectivity Graph) Given a set of cells Ci , Cj , Cf , Cl in the ground floor and a set of cells Cx, Cy,Cr, Ct the multidimensional connectivity graph refers to the multiple edges that connect the single floor cells and the above cells in the multi-floor space[2].

2.3 Shortest Path Algorithms

There are several algorithms that the accuration and time complexity have been tested to calculate the shortest distance between nodes on a graph, like Djikstra algorithm, Floyd-Warshall, Bellman-Ford, and Genetic Algorithm (GA). Here are the description of each algorithms.

2.3.1 Dijkstra's Algorithm

For each vertex within a graph we assign a label that determines the minimal length from the starting point s to other vertices v of the graph. In a computer we can do it by declaring an array $d[]$. The algorithm works sequentially, and in each step it tries to decrease the value of the label of the vertices. The algorithm stops when all vertices have been visited. The label at the starting point s is equal to zero ($d[s] = 0$); however, labels in other vertices v are equal to infinity ($d[v] = \infty$), which means that the length from the starting point s to other vertices is unknown. In a computer we can just use a very big number in order to represent infinity. In addition, for each vertex v we have to identify whether it has been visited or not. In order to do that, we declare an array of Boolean type called $u[v]$, where initially, all vertices are assigned as unvisited ($u[v] = false$). The Dijkstra's algorithm consists of n iterations. If all vertices have been visited, then the algorithm finishes. Otherwise, from the list of unvisited vertices we have to choose the vertex which has the minimum (smallest) value at its label (At the beginning, we will choose a starting point s). After that, we will consider all neighbors of this vertex (Neighbors of a vertex are those vertices that have common edges with the initial vertex). For each unvisited neighbor we will consider a new length, which is equal to the sum of the label's value at the initial vertex $v(d[v])$ and the length of edge l that connects them. If the resulting value is less than the value at the label, then we have to change the value in that label with the newly obtained value[8].

$$d[neighbors] = min(d[neighbors], d[v] + l) \tag{2.1}$$

After considering all of the neighbors, we will assign the initial vertex as visited ($u[v] = true$). After repeating this step n times, all vertices of the graph will be visited and the algorithm finishes or terminates. The vertices that are not connected with the starting point will remain by being assigned to infinity. In order to restore the shortest path from the starting point to other vertices, we need to identify array $p[]$, where for each vertex, where $v \neq s$, we will store the number of vertex $p[v]$, which penultimate vertices in the shortest path. In other words, a complete path from s to v is equal to the following statement.

$$P = (s, \ldots, p[p[p[v]]], p[p[v]], p[v], v) \tag{2.2}$$

Figure 2.6 is an implementation of Dijkstra's algorithm using the Java programming language.

```
1   Dijkstra(G, w, s) G-Graph, w-weight, s-source
2       for each vertex in Vertex[G]
3           dist[v] ← ∞
4           dist[s] ← 0
5   S ← empty set
6   while Vertex[G] is not an empty set
7       u ← Extract_Min(Vertex[G])
8       S ← S union {u}
9       for each edge (u,v) outgoing from u
10          if dist[u] + weight(u,v)< dist[v]
11              dist[v] ← dist[u] + weight(u,v)
```

Figure 2.6: Dijkstra's algorithm pseudocode

2.3.2 Floyd-Warshall Algorithm

Consider the graph G, where vertices were numbered from 1 to n. The notation dijk means the shortest path from i to j, which also passes through vertex k. Obviously if there is exists edge between vertices i and j it will be equal to d_{ij0}, otherwise it can assigned as infinity. However, for other values of dijk there can be two choices: (1) If the shortest path from i to j does not pass through the vertex k then value of d_{ijk} will be equal to d_{ijk-1}. (2) If the shortest path from i to j passes through the vertex k then first it goes from i to k, after that goes from k to j. In this case the value of d_{ijk} will be equal to $d_{ikk-1} + d_{kjk-1}$. And in order to determine the shortest path we just need to find the minimum of these two statements:

$$d_{ij0} = the\ length\ of\ edge\ between\ vertices\ i\ and\ j \tag{2.3}$$

$$d_{ijk} = min(d_{ijk-1}, d_{ikk-1} + d_{kjk-1}) \tag{2.4}$$

```
1   The Floyd-Warshall Algorithm
2   Floyd-Warshall (G,w,s) G-Graph, w-weight, s-source
3       init path[ ][ ];
4       for k = 1 to n
5           for each (i, j) in (1 ... n)
6               path[i][j] = min(path[i][j],
7               path[i][k]+path[k][j]);
8       end
```

Figure 2.7: Floyd-Warshall algorithm pseudocode

2.3.3 Bellman-Ford Algorithm

In comparison to Dijkstra's algorithm, the Bellman-Ford algorithm admits or acknowledges the edges with negative weights. That is why, a graph can contain cycles of negative weights, which will generate numerous number of paths from the starting point to the final destination, where each cycle will minimize the length of the shortest path. Taking into consideration this fact, let's assume that our graph does not contain cycles with negative weights. The array $d[]$ will store the minimal length from the starting point s to other vertices. The algorithm consists of several phases, where in each phase it needs to minimize the value of all edges by replacing $d[b]$ to following statement $d[a] + c; a$ and b are vertices of the graph, and c is the corresponding edge that connects them. And in order to calculate the length of all shortest paths in a graph it requires n–1 phases, but for those vertices of a graph that are unreachable, the value of elements of the array will remain by being assigned to infinity [8].

Figure 2.9 is the Bellman-Ford algorithm implementation using the Java programming language.

```
1  Bellman-Ford(G,w,s) G-Graph, w-weight, s-source
2  for each vertex ∈ Vertex[G]
3      dist[v] ← ∞
4      dist[s] ← 0
5  for i ← all vertices in G do
6      for each edge (u,v) ∈ Edge[G] do
7          Relax (u,v,w)
8      for each edge (u,v) ∈ Edge[G] do
9          if dist[v] > dist[u] + weight(u,v) then return false
10             return true
```

Figure 2.8: Bellman-Ford algorithm pseudocode

2.3.4 A* Algorithm

The A* (called as 'A-star') (Cvetanovic and Nofsinger 1990) is like other graph searching algorithms in that it can potentially search a huge area of the map. It's like Dijkstra's algorithm in that it can be used to find a shortest path. It's like breadth first search (BFS) in that it can use a heuristic to guide itself. In the simple case, it is as fast as BFS. Since in the worst case breadth-first search has to consider all paths to all possible nodes the time complexity of breadth-first search is increased depending on the number of nodes and edges in the graph [10].

This heuristic search ranks each node by an estimate of the best route that goes through that node. The typical formula is expressed as:

$$f(n) = g(n) + h(n) \tag{2.5}$$

where $f(n)$ is the score assigned to node n, this is the estimate of the best solution that goes through $n, g(n)$ is the actual cheapest cost of arriving at from the start node to n and $h(n)$ is the heuristic estimate of the cost to the goal from n (Fig. 7).

```
1  The AStar Algorithm
2  A*(G,s,w) G-Graph, w-weight, s-source
3  var closed = the empty set
4  var q = make_queue(path(start))
5  while q is not empty
6      var p = remove_first(q)
7      var x = the last node of p
8      if x in closed
9          continue
10     if x = goal
11         return p
12         add x to closed
13         foreach y in successors(p)
14             enqueue(q, y)
15 return failure
```

Figure 2.9: A* algorithm pseudocode

2.4 Three Dimensional Spaces

If a rubber ball thrown vertically from the floor, the ball will have a unique coordinate axes. Figure 2.10 is an illustration of the movement of a rubber ball thrown vertically. For example, a spherical coordinate to the axis of the two-dimensional space (in this case on the floor) is (5, 4) and the high ball as reflected by the floor is 2.5 cm, then the spherical coordinates into (5.4, 2.5). Each position of the vertical movement of the ball is a unique three dimensional space coordinates.

2.5 Distance between Two Points

Measuring the distance between two points on a three dimensional space can be illustrated in Figure 2.11 below. If there is a power outlet in an area of a wall in a room and there is an electric iron on the table, how many minimum length of electric cable that needed to connect the iron to a power outlet?

If the coordinates of the point P is (x_1, y_1, z_1) and point Q is (x_2, y_2, z_2), then the distance between point P and point Q or $|PQ|$ can be determined by using the following formula:

$$|PQ| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{2.6}$$

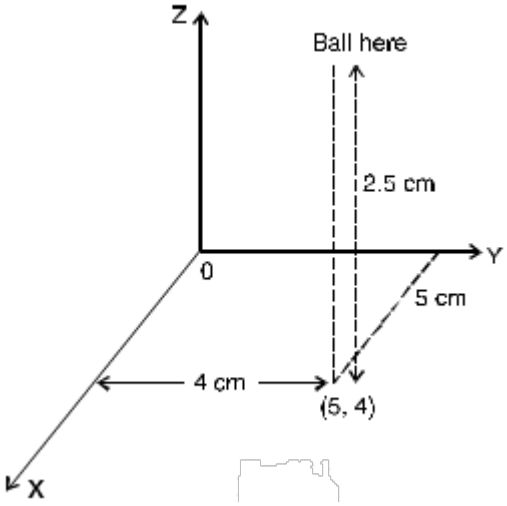

Figure 2.10: Illustration of Rubber Ball Movement

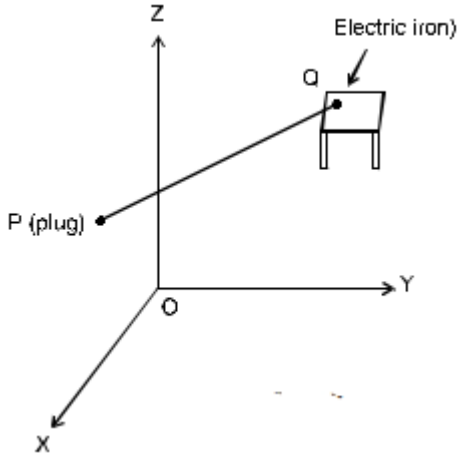

Figure 2.11: Ilustration of distance between two points on a Three Dimensional Spaces

Thus, it can be seen that the general equation of distance point $P(x, y, z)$ to the center point $O(0, 0, 0)$ will be as follows:

$$|PQ| = \sqrt{(x_2 - 0)^2 + (y_2 - 0)^2 + (z_2 - 0)^2} \tag{2.7}$$

$$|PQ| = \sqrt{(x_2)^2 + (y_2)^2 + (z_2)^2} \tag{2.8}$$

In other case, such as the distance between a room in the first floor and the one in the third floor of a bilding, we can't just use that formula to measure the distance. There is a route that we have to consider. Figure 2.12 shows the

route wich include rooms, corridors, and stairs of a building. To measure the distance between room R1 and room R2, we have to measure the exact length of the route.

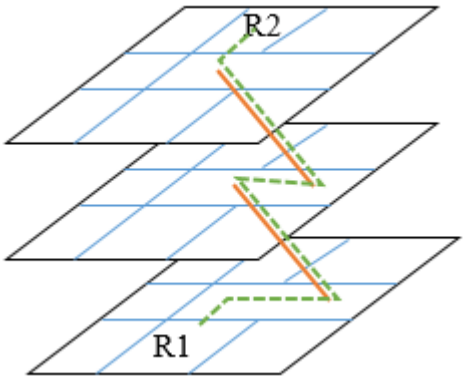

Figure 2.12: Ilustration of distance between two rooms

2.5.1 Spatial Data Distance Calculation using Haversine Formula

The Haversine formula is an equation important in navigation, giving great-circle distances between two points on a sphere from their longitudes and latitudes. These names follow from the fact that they are customarily written in terms of the haversine function, given by $haversin(\theta) = sin^2(\theta/2)$. The haversine formula is used to calculate the distance between two points on the Earth's surface specified in longitude and latitude [3].

$$d = 2rsin^{-}1(\sqrt{sin^2(\frac{\phi_2 - \phi_1}{2})} + cos(\phi_1)cos(\phi_2)sin^2(\frac{\psi_2 - \psi_1}{2})) \tag{2.9}$$

d is the distance between two points with longitude and latitude (ψ, θ) and r is the radius of the Earth.

2.6 Summary

This chapter is about the references that will be used in this final project. The theories and the algorithm will be implemented in building the Indoor Routing System.

Chapter III

System Methodology and Design

3.1 General Description

As the product of this final project, an application named TELUR IRIS (Telkom University Indoor Routing System) was made. This application is used for finding the shortest path between two rooms of Telkom University. Although, as declared in scope section of Chapter I, this application still use the data of D, E, and F Buildengs only. This application will show the user clearly about how the nodes of the graph represent the buildings by showing the nodes based on its three dimensional attribute. Several kinds of shortest path algorithm will be implemented and will be analyzed to get algorithm wich will give the best prformance that match this case.

3.2 System Design

3.2.1 The Data Structure

Dataset that used in this final project is the spatial data of Faculty of Informatics Telkom University, Bandung that include D, E, and F buildings. Figure 3.1 is the table relation of the dataset used in the TELUR IRIS Application. We need the data of buildings, rooms, corridors, stairs, and also Segments wich will be the relation or edges in the graph. For rooms, corridors, and stairs will have an attribute Building_ID as a foreign key that reference to Building_ID as the primary key of buildings. With this way, we know that each of rooms, corridors, and stairs is belongs to one of the buildings. However, for the corridor nodes that are outside the buildings, the Building_ID attribute will be set as null value. Segments table has source and destination attribute that actually filled with the nodes ID in graph.

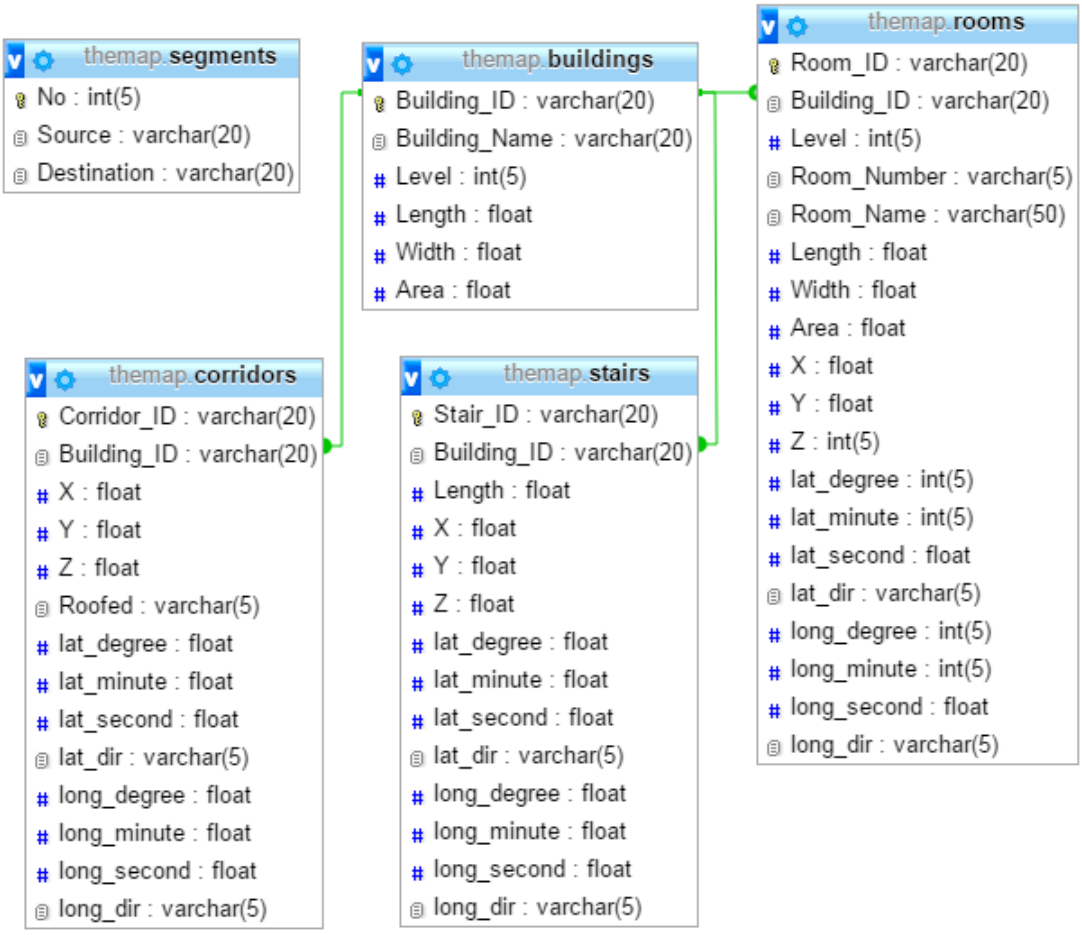

Figure 3.1: Relational table of dataset

3.2.2 System Flow

The main process of this application can be modeled with the flow chart below.

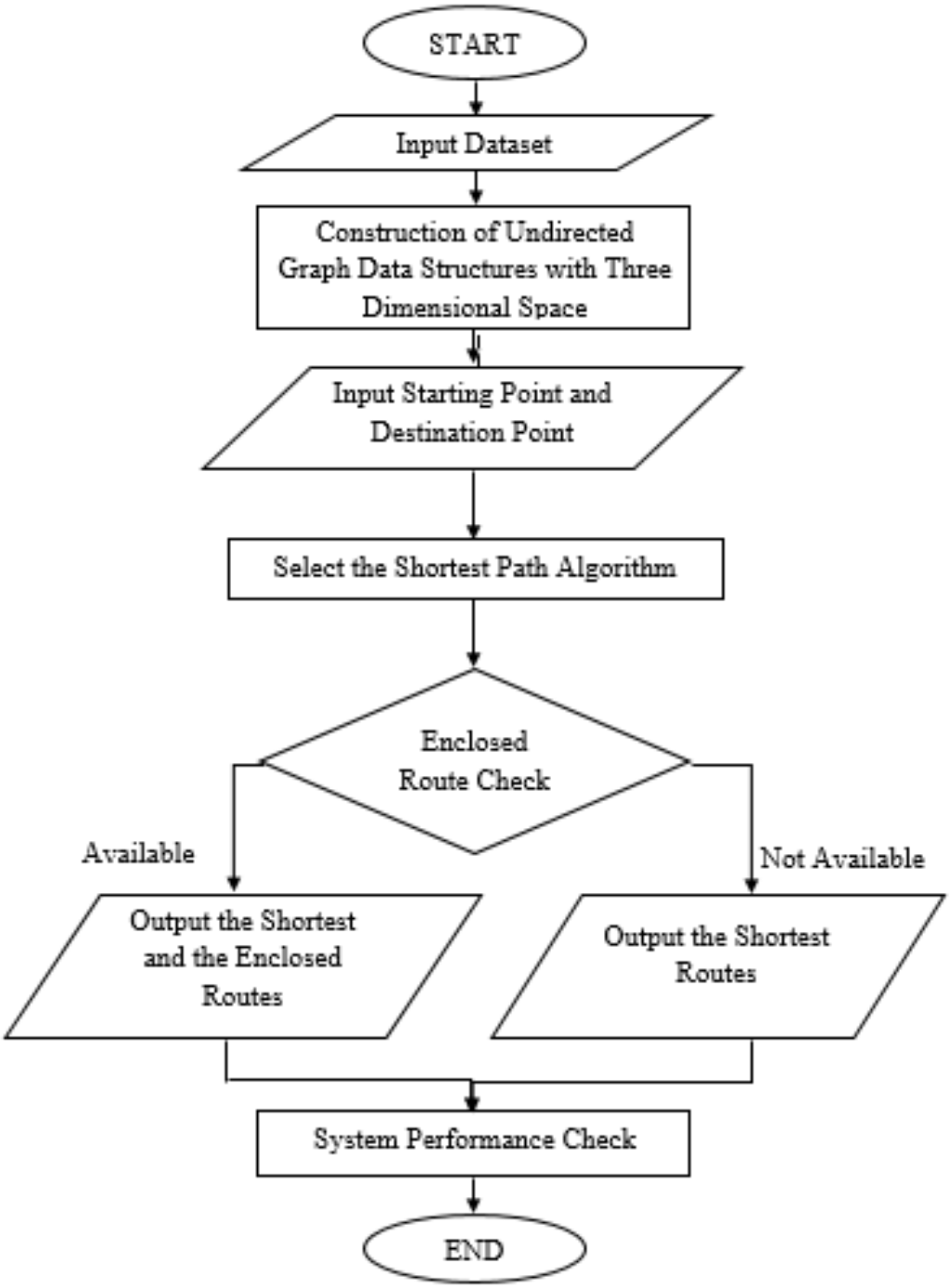

Figure 3.2: System Flow Chart

3.2.2.1 Construction of undirected Graph Data Structures with Three Dimensional Space

At this stage, the construction of undirected graph data structure using three dimensional space representation of the existing data sets. Unlike the indoor routing data representation that already described in Chapter 2, this will represent multi-floor building in three dimensions based on the location of the coordinates x, y, and z coordinates of altitude. This final project use the study case of D, E, and F buildings of School of Computing in Telkom University, Bandung, Indonesia. Figure 3.3 shows the area of Telokm University captured from Google Earth Application. The red rectanges are the D, E, and F buildings. Figure 3.4shows the zoomed in figure of D, E, and F buildings with the 2D graph modeling overlayed it. This is a way to get the longitude and latitude coordinates.

Figure 3.3: Telkom University area

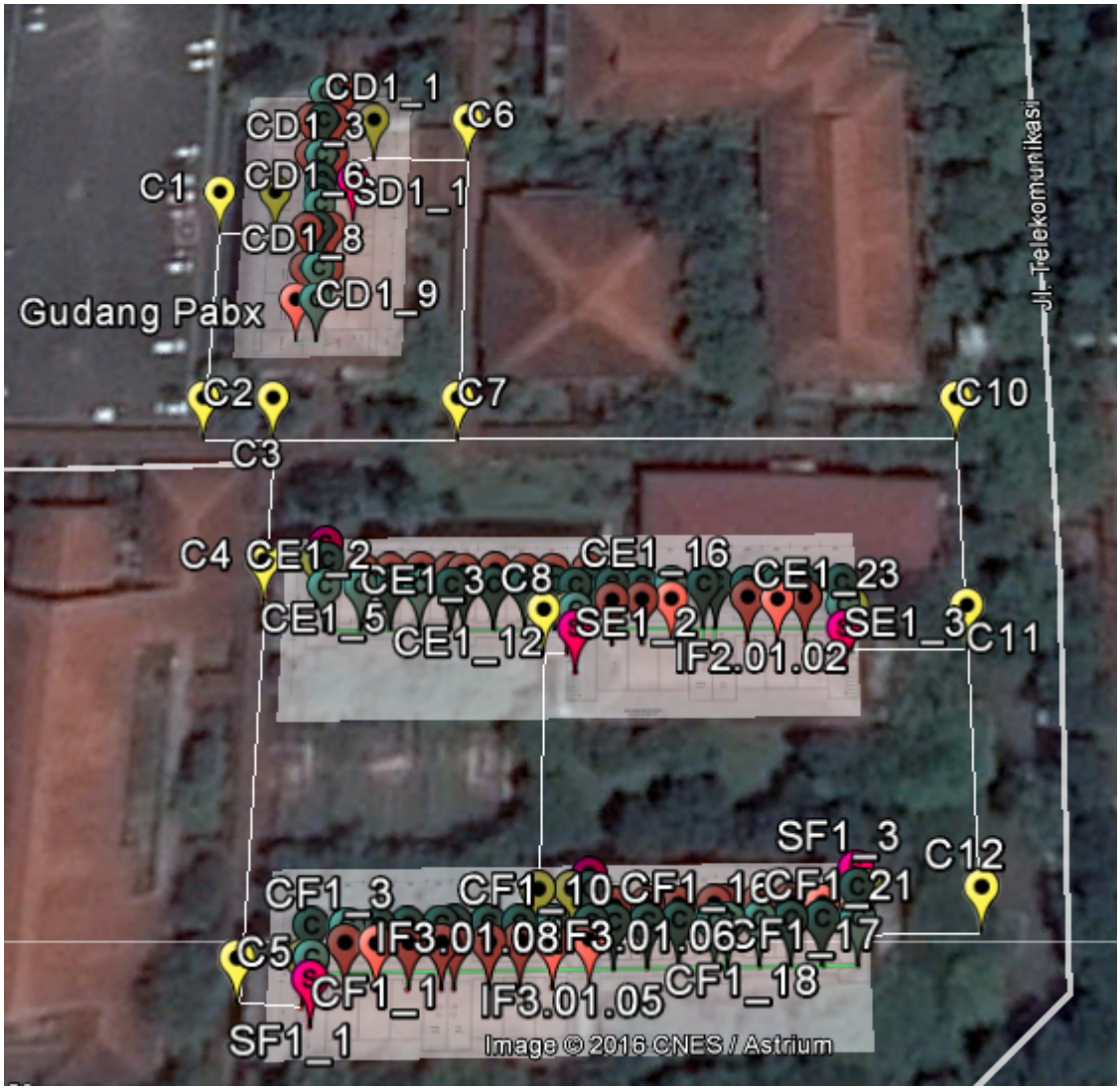

Figure 3.4: Telkom University area

In the next step, the 3D modelling can be implemented in the system. This system show the 3D model as 2D model like shown in Figure 3.5 because a building is actually an overlayed multi-objects. To show the 3D modelling, coordinate shifting method could be implemented. Coordinate shifting method will move the x and y coordinate of the nodes so it would not be overlayed anymore. Illustration of Three Dimensional Spaces model can be seen in Figure 3.6. red nodes mean the first level, green nodes mean the second floor, and blue ones mean the third floor.

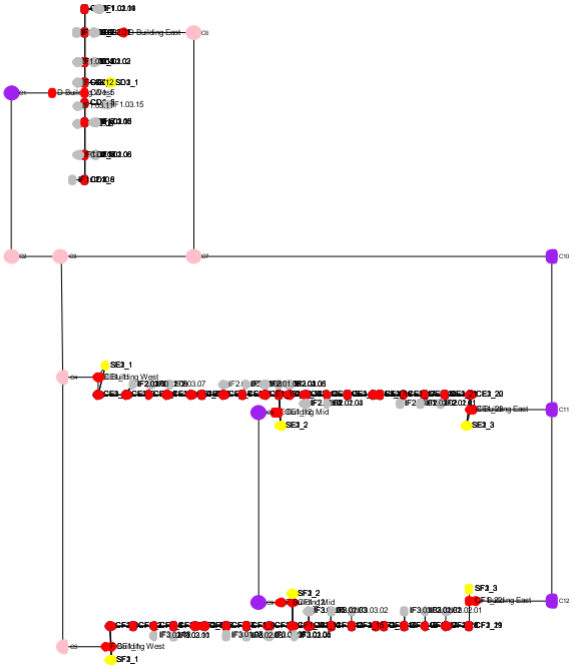

Figure 3.5: 2D system's graph modeling

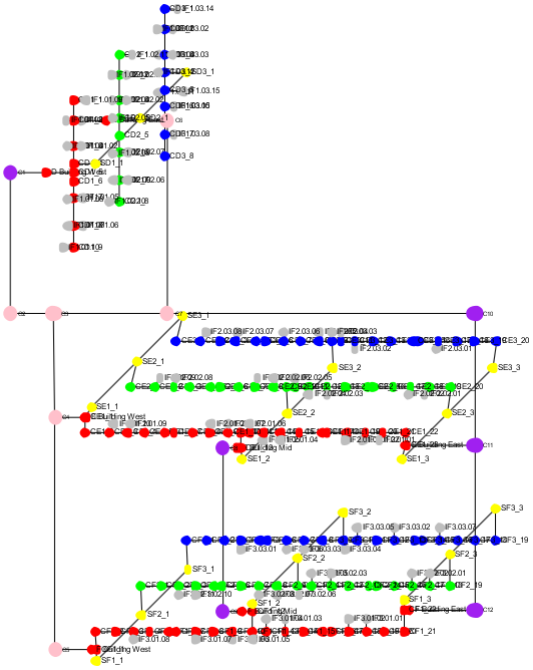

Figure 3.6: 3D system's graph modeling

3.2.2.2 Finding the Shortest Path

At this stage, shortest path algorithms will be implemented. There are Dijkstra's, Bellman-Ford, Floyd-Warshall, and A* algorithms that we could try to give the result of the shortest path. After the user choose one of the algorithms, shortest path searching will be done twice. The first search is using the full part of graph to show the shortest path weather it is open or close path. The second search will be eliminate nodes wich are the open space, so it will return a closed path.

By default, the system will show the close space path like the one shown in Figure 3.7. But, by clicking the shortest path text area, it could easily change the view to the shortest path like shown in Figure 3.8.

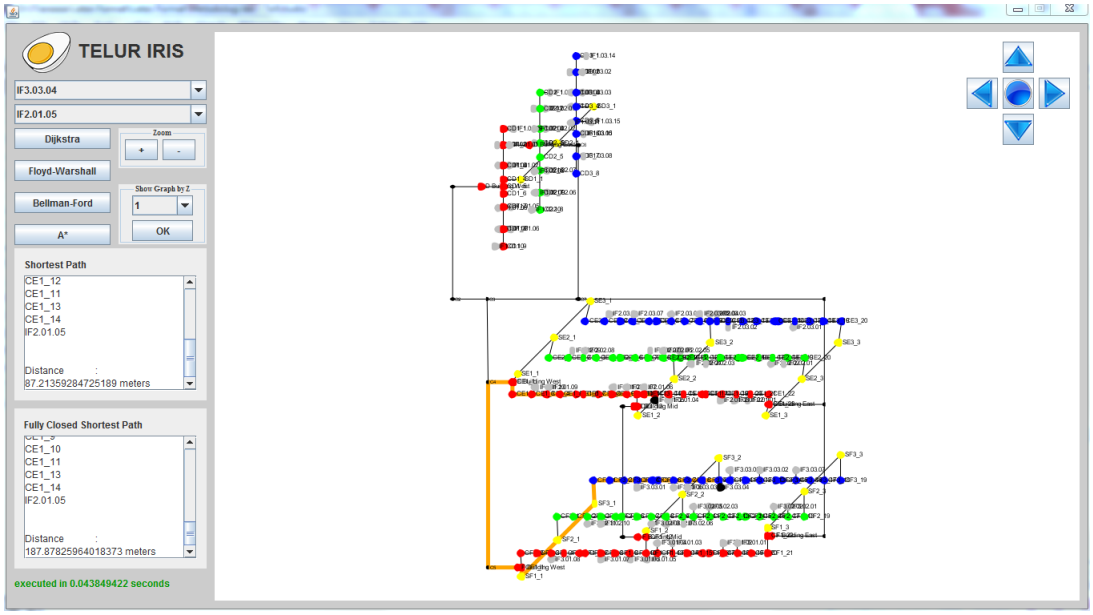

Figure 3.7: The close space shortest path

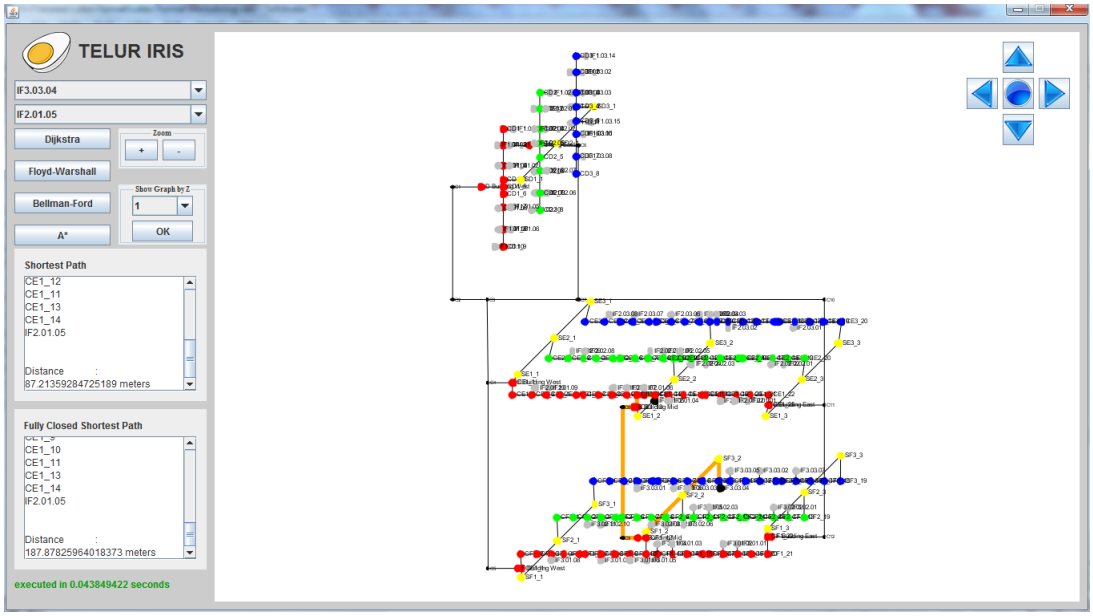

Figure 3.8: The shortest path

3.2.2.3 System Performance Check

After the algorithm return the shortest paths, time execution will be measured at this stage.

3.3 System Requirements Specification

In this final project, the hardware and software that will support routing indoor systems to be built and work properly are:

1. Hardware Specification
   Here is the following hardware specifications used in this final project.
   Model : Intel® Core™ i7-3770 CPU
   Prosesor : 3.40 GHz Intel Core i7
   RAM : 4 GB
   OS : Windows 7 Professional 64 bit

2. Software Specification
   Here is the following software specifications used in this final project.
   Application :

   - Java Netbeans IDE 8.0.2
   - JDK 1.8
   - MySQL

3.4 Summary

This chapter is about the plan of how to construct the Indoor Routing System using three dimensional spaces data structure. It also axplain the detail of the system flow and the system requirements.

Chapter IV

Testing and Analysis

4.1 System Testing

In this chapter, will be explain about testing and analysis that used in this final project. The result of this testing and analyzing process will show how good the perfomance of the system is. There are two aspects that will be the concern in this step, they are execution time and the correctness of the outputs.

4.1.1 Testing Objective

Based on the system model in chapter 3, the shortest path algorithms will be implemented in Indoor Routing System using three dimensional data structure. And the objectives of this testing process are:

1. To get the best system's performance by comparing the execution time between Dijkstra's, Bellman-Ford, Floyd-Warshall, and A* algorithms.

2. To get the system's accuracy by comparing the distance as the result of TELUR IRIS system and the distance in real life.

4.1.2 Testing Scenario

The following statements are the scenario of testing process:

1. **Testing of system's perfomance in execution time**. In this section, each of dijkstra's, bellman-ford, floyd-warshall, and A* will be tested five times to get the execution time. After that the average amount can be assumed as the exact execution time. this testing will be implemented in both open and close space wich is has different count of nodes and edges.

2. **Testing of system's accuracy**. In this section, comparison between the distance as the result of TELUR IRIS system and the distance in real life will be calculated. For each scenario, there will be three samples of calculation result.

The scenario of this test will be devided into 4, those are:

- Distance calculation between two rooms that have same Building_ID and same z attribute.
- Distance calculation between two rooms that have same Building_ID but different z attribute.
- Distance calculation between two rooms that have different Building_ID but same z attribute.
- Distance calculation between two rooms that have different Building_ID and different z attribute.

4.2 Testing Result

4.2.1 Time Execution Comparison

Here are the result of time execution testing. For the testing sample, the shortest path between node IF1.01.01 to node IF3.03.07 were choosen as a sample. This test implemented Dijkstra, Floyd-Warshall, Bellman-Ford, and A* Algorithms to search that shortest path. The testing do the searching process five times for each algorithms and it took the average value of the execution times as the result. All the time values are in seconds.

The output of five times Dijkstra's Algorithm running:
*****dijkstra algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time dijkstra : 0.176570559
*****dijkstra algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time dijkstra_close : 0.059495901
executed in 0.23606646 seconds

*****dijkstra algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time dijkstra : 0.059805929
*****dijkstra algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time dijkstra_close : 0.030269795
executed in 0.090075724 seconds

*****dijkstra algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time dijkstra : 0.048037625
*****dijkstra algorithm node IF1.01.01 to IF3.03.07*****

graph.getNodeCount() = 266
execution time dijkstra_close : 0.035783754
executed in 0.083821379 seconds

*****dijkstra algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time dijkstra : 0.04465599
*****dijkstra algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time dijkstra_close : 0.030038336
executed in 0.074694326 seconds

*****dijkstra algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time dijkstra : 0.06649877
*****dijkstra algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time dijkstra_close : 0.032153671
executed in 0.098652441 seconds

The output of five times Floyd-Warshall Algorithm running:
*****Floyd-Warshal algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time fw : 0.280716078
*****Floyd-Warshal algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time fw_close : 0.175645076
executed in 0.456361154 seconds

*****Floyd-Warshal algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time fw : 0.262804205
*****Floyd-Warshal algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time fw_close : 0.100703012
executed in 0.363507217 seconds

*****Floyd-Warshal algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time fw : 0.295337272
*****Floyd-Warshal algorithm node IF1.01.01 to IF3.03.07*****

graph.getNodeCount() = 266
execution time fw_close : 0.081183668
executed in 0.37652094 seconds

*****Floyd-Warshal algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time fw : 0.283726813
*****Floyd-Warshal algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time fw_close : 0.238191704
executed in 0.521918517 seconds

*****Floyd-Warshal algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time fw : 0.282648442
*****Floyd-Warshal algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266

execution time fw_close : 0.263365865
executed in 0.546014307 seconds

The output of five times Bellman-Ford Algorithm running:
****Bellman-Ford algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time bf : 0.337928182
****Bellman-Ford algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time bf_close : 0.188902291
executed in 0.526830473 seconds

****Bellman-Ford algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time bf : 0.221083213
****Bellman-Ford algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time bf_close : 0.234484116
executed in 0.455567329 seconds

****Bellman-Ford algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272

execution time bf : 0.29284962
****Bellman-Ford algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time bf_close : 0.240597956
executed in 0.533447576 seconds

****Bellman-Ford algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time bf : 0.273845218
****Bellman-Ford algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time bf_close : 0.246817972
executed in 0.52066319 seconds

****Bellman-Ford algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time bf : 0.28561423
****Bellman-Ford algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time bf_close : 0.236419664
executed in 0.522033894 seconds

The output of five times A* Algorithm running:
****A* algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time astar : 0.065852526
****A* algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time astar_close : 0.016076835
executed in 0.081929361 seconds

****A* algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time astar : 0.03272878
****A* algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time astar_close : 0.023422293
executed in 0.056151073 seconds

****A* algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272

execution time astar : 0.030558941
****A* algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time astar_close : 0.01713822
executed in 0.047697161 seconds

****A* algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time astar : 0.026007271
****A* algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time astar_close : 0.022123787
executed in 0.048131058 seconds

****A* algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 272
execution time astar : 0.036428937
****A* algorithm node IF1.01.01 to IF3.03.07*****
graph.getNodeCount() = 266
execution time astar_close : 0.032021663
executed in 0.0684506 seconds

The tables below shows the clearer comparison according to the console output of the system above. Table 4.1 shows the comparison of the shortest path algorithms in showing the actual shortest path where all nodes (total 272 nodes) of the graph are included in the algorithm's calculation. Table 4.2 shows the comparison of the shortest path algorithms in showing the shortest path in close space where nodes wich are the open space will not be calculated in shortest path algorithm (total 266 nodes).

Table 4.1: Shortest path execution time comparison for Whole Space

Itteration	**Dijkstra**	**Floyd-Warshall**	**Bellman-Ford**	**A***
1	0.176570559	0.280716078	0.337928182	0.065852526
2	0.059805929	0.262804205	0.221083213	0.03272878
3	0.048037625	0.295337272	0.29284962	0.030558941
4	0.04465599	0.283726813	0.273845218	0.026007271
5	0.06649877	0.282648442	0.28561423	0.036428937
AVERAGE	0.079113775	0.281046562	0.282264093	0.038315291

Table 4.2: Shortest path execution time comparison for Close Space

Itteration	Dijkstra	Floyd-Warshall	Bellman-Ford	A*
1	0.059495901	0.175645076	0.188902291	0.016076835
2	0.030269795	0.100703012	0.234484116	0.023422293
3	0.035783754	0.081183668	0.240597956	0.01713822
4	0.030038336	0.238191704	0.246817972	0.022123787
5	0.032153671	0.263365865	0.236419664	0.032021663
AVERAGE	0.037548291	0.171817865	0.2294444	0.02215656

And since the system will give both open and close space shortest paths, the comparison of system's time execution will be the total value of both situation. The value are shown in Table 4.3.

Table 4.3: System's execution time comparison

Itteration	Dijkstra	Floyd-Warshall	Bellman-Ford	A*
1	0.23606646	0.456361154	0.526830473	0.081929361
2	0.090075724	0.363507217	0.455567329	0.056151073
3	0.083821379	0.37652094	0.533447576	0.047697161
4	0.074694326	0.521918517	0.52066319	0.048131058
5	0.098652441	0.546014307	0.522033894	0.0684506
AVERAGE	0.116662066	0.452864427	0.511708492	0.02215656

According to the data above, pictures below are the chart figures that show how far are the execution time deviation between each shortest path algorithms.

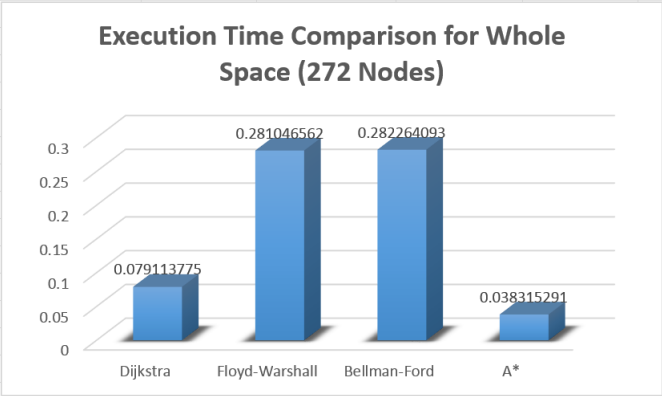

Figure 4.1: Execution Time Comparison for Whole Space

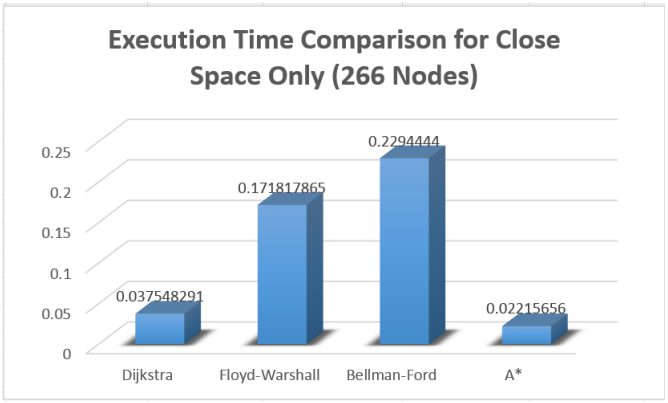

Figure 4.2: Execution Time Comparison for Close Space Only

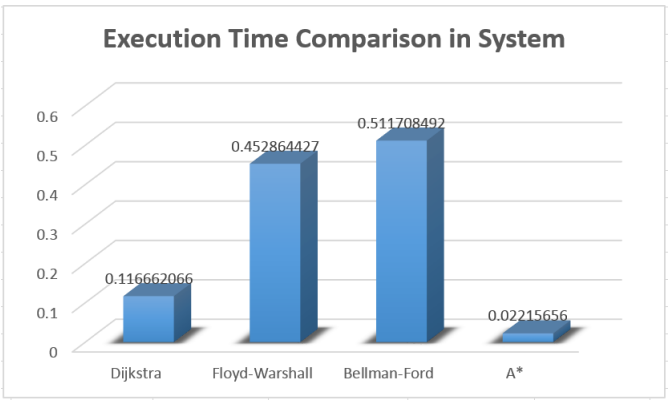

Figure 4.3: Execution Time Comparison in System

As seen in the charts above, it could be concluded that A* Algorithm give the smallest execution time wich mean A* algorithm is the best algorithm to implement in this case besides Dijkstra, Floyd-Warshall, or Bellman-Ford algorithm.

4.2.2 Distance Calculation Comparison

Here are the result of the system's accuracy testing. This testing will show the deviation between the result of system calculation and the real distance as shown in Table 4.4. All the distances are in meters.

Table 4.4: Distance deviation between system result and in the real life

No.	Source	Destination	SPD_S	SPD_R	Deviation ($ABS\,(SP\_S - SP\_R)$)
1	IF3.03.04	IF3.03.05	8.661221996	6	2.661221996
2	IF3.02.05	IF3.02.01	34.78388051	34.7	0.083880508
3	IF3.01.02	IF3.01.03	21.56047226	21.7	0.139527743
4	IF3.03.03	IF3.02.05	38.7783564	34.1	4.6783564
5	IF3.03.03	IF3.01.05	41.18405404	35.4	5.784054043
6	IF2.01.10	IF2.02.09	36.66114963	31	5.661149626
7	IF2.01.10	IF3.01.08	102.1953267	89.4	12.79532672

Figure 4.4 shows the fluctuation of the testing deviation result. From the chart, we can see that the daviation result is quiet fluctuate. This result means that this system has a deficiency in resuting the distance with average deviation velue is 4.543359576 meters.

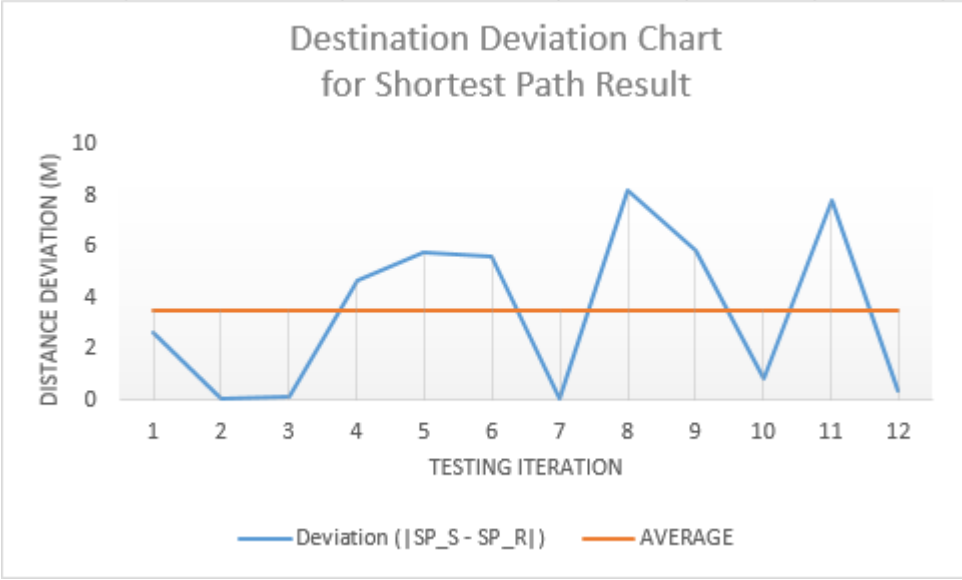

Figure 4.4: Distance deviation testing result

4.3 Summary

Chapter V

Conclusions and Recommendations

5.1 Conclusions

After the implementation, testing, and analyzing process have been done, it could be concluded that:

1. Indoor Routing System could be implemented using three dimensional spaces.
2. The most suitable Shortest Path Algorithm to the three dimensional spaces data between Dijkstra, Floyd-Warshall, Bellman-Ford, and A* Algorithms is A* Algorithm that give the smallest value of execution time.

5.2 Future Work

Since this final project have been done, there are a few recommendations to make this project better, those are:

1. Add some more dataset until the whole area of Telkom University. After that, it could be published in college's website so this system could be used for everyone.
2. Find another better way to get the exact longitude and latitude for the points. One of the examples is take a long walk from points to points and record the Longitude and Latitude coordinates.
3. It would be better if the graph could be viewed with the 3D blueprints of the buildings.

Bibliography

[1] Afyouni, I., Cyril, R., and Christophe, C. Spatial models for context-aware indoor navigation systems: A survey. *Journal of Spatial Information Science 1*, 4 (2012), 85–123.

[2] Alamri, S. M. *Adjacency-based indexing for moving objects in spatial databases*. PhD thesis, Monash University. Faculty of Information Technology. Clayton School of Information Technology, 2014.

[3] Chopde, N. R., and Nichat, M. Landmark based shortest path detection by using a* and haversine formula. *International Journal of Innovative Research in Computer and Communication Engineering 1*, 2 (2013), 298–302.

[4] Gotlib, D., Gnat, M., and Marciniak, J. The research on cartographical indoor presentation and indoor route modeling for navigation applications. In *Indoor Positioning and Indoor Navigation (IPIN), 2012 International Conference on* (2012), IEEE, pp. 1–7.

[5] Güting, R. H. An introduction to spatial database systems. *The VLDB Journal—The International Journal on Very Large Data Bases 3*, 4 (1994), 357–399.

[6] Han, L., Zhang, T., and Wang, Z. The design and development of indoor 3d routing system. *Journal of Software 9*, 5 (2014), 1223–1228.

[7] Karimi, H. A. *Universal navigation on smartphones*. Springer Science & Business Media, 2011.

[8] Magzhan, K., and Jani, H. M. A review and evaluations of shortest path algorithms. *International journal of scientific & technology research 2*, 6 (2013).

[9] Okabe, A., Boots, B., Sugihara, K., and Chiu, S. N. *Spatial tessellations: concepts and applications of Voronoi diagrams*, vol. 501. John Wiley & Sons, 2009.

[10] Sathyaraj, B. M., Jain, L. C., Finn, A., and Drake, S. Multiple uavs path planning algorithms: a comparative study. *Fuzzy Optimization and Decision Making 7*, 3 (2008), 257–267.

Lampiran