

Department of Electrical and Computer Engineering

**“AccessBIM” - A Model of Environmental Characteristics for
Vision Impaired Indoor Navigation and Way Finding**

Jayakody Arachchilage Don Chaminda Anuradha Jayakody

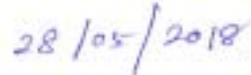
**This thesis is presented for the Degree of
Doctor of Philosophy
of
Curtin University**

June 2018

Declaration

To the best of my knowledge and belief, this thesis contains no material previously published by any other person except where due acknowledgment has been made.

This thesis contains no material, which has been accepted for the award of any other degree or diploma in any university.



J. A. D. C. A. Jayakody

Date

Parts of this thesis has been previously published as listed below.

Conference papers:

- J. A. D. C. A. Jayakody, I. Murray, J. Herrmann, S. Lokuliyana, V. R. Dunuwila (in press) “Enhanced Algorithmic Implementation to Assist Real-time Indoor Map Generation for Vision Impaired Individuals,” in the 13th International Conference on Computer Science & Education (ICCSE), Colombo 2018.
- J. A. D. C. A. Jayakody, I. Murray, J. Herrmann, S. Lokuliyana, V. R. Dunuwila (in press) “Tempcache: A Database Optimization Algorithm for Real-time Data handling in Indoor Spatial Environments,” in the 13th International Conference on Computer Science & Education (ICCSE), Colombo 2018.
- J. A. D. C. A. Jayakody, S. Lokuliyana, I. Murray, J. Hermann, N. Gamage, "Optimized AccessBIM Model to Assist Vision Impaired Individuals with Real-time Indoor Map," in First One Curtin International Postgraduate Conference, Malaysia, 2017.
- J. A. D. C. A. Jayakody, I. Murray, J. Herrmann, D. S. A Kandawela, S. E. C. Nanayakkara, S. Lokuliyana, “A Database Optimization Model with Quantitative

Benchmark,” 2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT), India, 2016.

- J. A. D. C. A. Jayakody, S. Lokuliyana, K.D.D.Chathurangi, D.R.Vithana, "Indoor Positioning: Novel approach for Bluetooth Networks using RSSI Smoothing," International Journal of Computer Applications, vol. 137, (13), 2016.
- J. A. D. C. A. Jayakody, I. Murray, J. Herrmann, “An Algorithm for Labeling Topological Maps to Represent Point of Interest for Vision Impaired Navigation,” in International Conference on Indoor Positioning and Indoor Navigation (IPIN), Canada, 2015.
- J. A. D. C. A. Jayakody, I. Murray, J. Herrmann, “Database modeling for vision impaired indoor navigation systems: Extended Abstract,” in International Conference on Advances in ICT for Emerging Regions (ICTER), Colombo, 2015.
- J. A. D. C. A. Jayakody, I. Murray, “The Construction of an Indoor Floor Plan Using a Smartphone for Future Usage of Blind Indoor Navigation,” in International Conference on Contemporary Computing and Informatics, Bangalore, 2014.
- J. A. D. C. A. Jayakody, I. Murray, “Proposed Novel Schema Design for Map Generation to Assist Vision Impaired in an Indoor Navigation Environment,” in International Conference on Contemporary Computing and Informatics, Bangalore, 2014.
- J. A. D. C. A. Jayakody, I. Murray, “Proposed Methodology for Labeling Topological Maps to Represent Rich Semantic Information for Vision Navigation,” in International Conference on Indoor Positioning and Indoor Navigation, France, 2013.
- J. A. D. C. A. Jayakody, L. Rupasinghe, K. A. M. S. Perera, H. H. P. M. Herath, T. M. A. Thennakoon, S. U.Premanath, "The development of the CityGML GeoBIM extension for Real-Time assessable model (Integration of BIM and GIS)," in the Second National Conference on Technology and Management, Sri Lanka, 2013.
- J. A. D. C. A. Jayakody, E.D.S. Nishani, U.M.R.Senaratne, "Spatial model for indoor

unknown infrastructure with built information," in SAITM Research Symposium on Engineering Advancements, Sri Lanka, 2013.

- J. A. D. C. A. Jayakody, I. Murray, "An AccessBIM model for environmental characteristics for vision impaired indoor navigation and way finding," in International Conference on Indoor Positioning and Indoor Navigation, Kensington, 2012.

Dedication

To my family, who have supported me throughout this work and thereby made it possible. Especially my two daughters (Sayuri and Sasanya) and my wife who has had the patience to accept my seemingly never-ending late nights in front of the computer, and to my mother who shouldered many burdens so that I could be free to carry on this work.

Also to my supervisors, whose constant questions and feedback have shaped this thesis into something far better than what it would have been without their input.

Abstract

The navigation of indoor and outdoor environments play a pivotal role in the daily routine of humans. Navigation systems that provide path planning and exploration services for outdoor environments are readily available while navigation within a building is still a challenge due to limited information availability and the poor quality of GPS signals, which makes it difficult to capture characteristics within the indoor environment. Consequently, the use of GPS tracking devices for real-time map generation is not feasible. Indoor navigation is particularly difficult for people with vision impairment. According to the factsheet of the World Health Organization (WHO) as of October 2017, over 253 million people are estimated to be vision impaired: 36 million to be blind, and 217 to have poor vision. Currently, most blind and vision-impaired individuals use the white cane as an assistive tool and are often accompanied by care takers or voluntary helpers.

Most modern indoor environments consist of complex architectural structures with varying arrangement of physical objects. Since retrieving indoor location information has been challenging for the vision impaired, it would be helpful if spatial information of doors, walls and staircases were made available.

To address the above-mentioned problem, this thesis presents an improved schema design, an Accessible Building Information Model (AccessBIM) which could be used for generating an indoor map that could instruct vision impaired individuals in navigation, by the classification of real world objects and their locations. AccessBIM is a real-time relational database, which acts as the main component of the central system implemented to manipulate crowdsourced data such as the floor plan and architectural data along with semantic information within the built environment. The AccessBIM database stores information on the indoor arrangement of objects within buildings to facilitate the exchange and interoperability of real-time information. The database is equipped with an optimization algorithm that reduces the query execution time with the support of indexing, query re-writing, schema redesigning and a memory optimization technique introduced as “BIMcache”.

In order to create a real-time map, the AccessBIM manipulates crowdsourced data from “smart devices” or AccessBIM users. The collection and storage of crowdsourced data, database optimization, API functions and the map construction algorithms were tested using a simulated test engine.

The AccessBIM framework has the potential to play an integral role in assistive technologies related to localization and mapping, thus significantly improving the independence and quality of life for people with vision impairment whilst also decreasing the cost to the community related to support workers.

Acknowledgements

First and Foremost, I would like to express my sincere gratitude to my supervisor, Associate Prof. Iain Murray for the continuous support provided for my Ph.D. study and research, and for his patience, motivation, enthusiasm, immense knowledge and his guidance which helped me at all times of the research and in writing thesis. I could not have imagined having a better advisor and mentor for my Ph.D. study.

Next gratitude goes to my co-supervisor; Dr. Hannes Herrmann who provided constrictive feedback on the direction of the thesis and papers co-authored.

Besides my superiors, I would like to thank the rest of my thesis committee chair person Dr. Cesar Ortega-Sanchez and his committee members for the encouragement, insightful comments, and hard questions.

My sincere thanks go to Prof. Lalith Gamage for giving me this treasured opportunity to continue my doctoral study in a prestigious university such as Curtin, Australia. Further, I would be grateful to mention the mentors working under Information Systems Engineering Department at Sri Lanka Institute of Information Technology, Prof. Samantha Thelijjagoda, Prof. Koliya Pulasinghe, Dr. Malitha Wijesundara, and Dr. Pradeep Abegunawardena who mentored and advised me towards this endeavor.

I thank my fellow peers Dr. Nimsiri Abayasinghe, Mr. Lakmal Rupasinghe, Ms. Nimali Rajakaruna, Ms. Nimalika Fernando and Mr. Dhammika De Silva of the Indoor Navigation research group, for the stimulating discussions, sleepless nights we worked together before deadlines, and for all the fun we have had in the last six years. Also, I thank my friends at Sri Lanka Institute of Information Technology those who supported me throughout my research work: Ms. Shashika Lokuliyana, Ms. Sachini Kandawala and my research assistants Ms. Narmada Gamage and Ms. Vandhana Dunuwila.

Finally, I would like to acknowledge my family, specially my parents Mr. S. Jayakody and Mrs. P. Jayakody, for giving birth to me at the first place and supporting me spiritually throughout my life.

Last but not the least, I would like to thank my wife Madushani Jayakody who stood by my side at tough times in my life and my two daughters, Sayuri Jayakody and Sasanya Jayakody for their unconditional love.

Table of Contents

Declaration	i
Dedication	iv
Abstract	v
Acknowledgements	vii
Table of Contents	ix
List of Figures	xiv
List of Tables.....	xx
List of Equations	xxii
1 Chapter 1 – Introduction	1
1.1 Problem Overview.....	1
1.2 Motivation for indoor map generation	2
1.3 Motivation for database optimization	3
1.4 Research Gap.....	3
1.5 Problem Statement	4
1.6 Contribution and Significance.....	5
1.7 Assumptions	7
1.8 Thesis Outline	8
2 Chapter 2 – Background and Literature Review.....	9
2.1 Chapter Overview	9

2.2 Vision Impaired Mobility.....	9
2.2.1 Social Aspect of Vision Impaired Mobility.....	9
2.2.2 Technical Aspect of Vision Impaired Mobility.....	13
2.3 Indoor Navigation and Map Generation.....	18
2.3.1 Algorithms and Frameworks.....	18
2.3.2 Device Based Solutions.....	19
2.3.3 Mobile Application Based Solutions.....	20
2.4 Real-time Databases and Optimization.....	21
2.4.1 General Concept.....	21
2.4.2 Real-time Databases.....	22
2.5 Object-Relational Implementation.....	22
2.6 Database Optimization and Performance Tuning.....	24
2.7 Contribution to the Research Project.....	25
2.8 Chapter Conclusion.....	28
3 Chapter 3 – Framework, Database and Test Environment Design and Implementation.....	30
3.1 Chapter Overview.....	30
3.3 Implementation Phases of the Research Work.....	31
3.2 Experimental Design Objective.....	30
3.3.1 Methodology of Framework Implementation.....	31
3.3.2 Database Designing Approach.....	38

3.3.3 Test Environment Implementation Approach	57
3.4 Real-Time Map Creation.....	65
3.5 Chapter Conclusion.....	67
4 Chapter 4 – Map Construction Algorithms.....	69
4.1 Chapter Overview	69
4.2 The Four Algorithms.....	69
4.3 Algorithm for Indoor Real-time Map Generation.....	70
4.4 Algorithm for Database Optimization.....	74
4.5 Algorithm for BIMcache Optimization.....	75
4.6 Algorithm for Crowdsourced Data Collection.....	78
4.7 Chapter Conclusion.....	79
5 Chapter 5 – The Simulation Environment and Test cases	80
5.1 Chapter Overview	80
5.2 The Simulation Environment	80
5.3 Test Scenarios	80
5.4 Chapter conclusion.....	98
6 Chapter 6 – Evaluation and Discussion	100
6.1 Chapter Overview	100
6.2 Evaluation of the AccessBIM framework.....	100
6.3 Database optimization.....	102

6.3.1 Performance Enhancement with Stored Procedures and Indexing For Real-Time Indoor Map Generation	103
6.3.2 Performance Enhancement with the use of BIMcache for Real-Time Indoor Map Generation	121
6.3.3 Performance Enhancement with the use of Query Rewriting for Real-Time Indoor Map Generation	125
6.4 Comparison of Results and Discussion.....	126
6.4.1 Discussion on stored procedures and indexing.....	127
6.4.2 Discussion on the use of BIMcache	129
6.4.3 Discussion on query rewriting.....	130
6.4.4 Comparison of the total time in real-time map generation.....	131
6.4.5 Quantitative benchmark for performance enhancement.....	136
6.5 Evaluation of the AccessBIM framework against existing work.....	143
6.5.1 Comparison between query execution time and size of data retrieved	143
6.5.2 Comparison between the frequency of executing the query and the number of users	148
6.6 Chapter conclusion.....	151
7 Chapter 7 – Conclusion and Future Research.....	153
7.1 Chapter Overview	153
7.2 Research outcomes and significance.....	153
7.3 Recommended framework	157
7.3.1 Recommended database optimization model	158

7.3.2 Recommended real-time map generation model	159
7.4 Limitations of the research	160
7.5 Future Research Work.....	160
References.....	162
Appendices.....	174
Appendix A: System Specification	174
Appendix B: C# DLL to interact C# application and web service.....	175
Appendix C: 7 th floor of Sri Lanka Institute of Information Technology.....	184
Appendix D: Results of query execution	188
Appendix E: Summary of the Interview conducted with the Visual Impaired of Employee Federation of Ceylon.....	197
Appendix F: List of Available Stored Procedures	199

List of Figures

Figure 1.1: Research component diagram for AccessBIM model.....	7
Figure 2.1: Laser Cane [30].	144
Figure 2.2: Sonic path finder [32]......	144
Figure 2.3: Handheld mobility device [33]......	155
Figure 2.4: Sample GPS tracker [34]......	155
Figure 2.5: Four-quadrant view of the database world [53]	23
Figure 3.1: Main phases of AccessBIM model Implementation	331
Figure 3.2: AccessBIM Framework	33
Figure 3.3: Collecting data through GAIT Analysis	35
Figure 3.4: User Interface Layer.....	36
Figure 3.5: Process of functional layer	37
Figure 3.6: ER Diagram with Table relation	39
Figure 3.7: Schema Design with relations.....	40
Figure 3.8: Method of calculating angle.....	46
Figure 3.9: Overview of BIMcache [64]	52
Figure 3.10: Output results of BIMcache	53
Figure 3.11: Output of query execution without SQL JOIN	55
Figure 3.12: Output of query execution with SQL JOIN	55
Figure 3.13: Created simulator (Virtual environment).....	60

Figure 3.14: User starts navigation.....	60
Figure 3.15: User navigation and data collection	61
Figure 3.16: User finish navigation and data collection	61
Figure 3.17: Overview of simulated test Engine	62
Figure 3.18: Identify user’s angle.....	63
Figure 3.19: Obstacle detection	64
Figure 3.20: Identify non-movable obstacle.....	65
Figure 3.21: Overview of map generation.....	66
Figure 3.22: Generated real-time map according to multiple user navigations.....	67
Figure 4.1: Dynamic Indoor Real-Time Map Generation	71
Figure 4.2: Flow of AccessBIM database optimization model	73
Figure 4.3: AccessBIM Database Optimizer Algorithm	74
Figure 4.4: Flow of BIMcache time reduction model	76
Figure 4.5: AccessBIM BIMcache Optimizer Algorithm	77
Figure 4.6: Crowdsourced Data Gathering.....	78
Figure 5.1: Floor plan of simulated environment (7 th floor of Sri Lanka Institute of Information Technology).....	881
Figure 5.2: Desired navigation plan for scenario 1.....	83
Figure 5.3: Wall_info table.....	84
Figure 5.4: Door_info table	84

Figure 5.5: Object_info table.....	85
Figure 5.6: Simulator with object positioning.....	85
Figure 5.7: Update movement_info table.....	86
Figure 5.8: Generated real-time map according to navigation.....	87
Figure 5.9: Desired navigation plan for scenario 2.....	88
Figure 5.10: Movement_info Table after test case 1.....	89
Figure 5.11: Simulator after changing object's location.....	89
Figure 5.12: Updated real-time map with new object location.....	90
Figure 5.13: Desired navigation plan for scenario 3.....	91
Figure 5.14: Updated real-time map with new environmental characteristics.....	92
Figure 5.15: Desired navigation plan for scenario 4.....	93
Figure 5.16: Multiple user navigations at once.....	94
Figure 5.17: Updated map with multiple user navigations.....	95
Figure 5.18: Desired navigation plan for test case 5.....	96
Figure 5.19: Door closed in simulator.....	97
Figure 5.20: Updated map with current door status and routes.....	98
Figure 6.1: Flow of BIMcache () stored procedure.....	105
Figure 6.2: Cost comparison for stored procedure with indexing.....	106
Figure 6.3: Flow of select_starting_and_ending () stored procedure.....	107
Figure 6.4: Cost comparison of select_starting_and_ending () stored procedure.....	108

Figure 6.5: Flow of select_values_from_wall_info () stored procedure	109
Figure 6.6: Cost comparison for select_values_from_wall_info stored procedure	110
Figure 6.7: Flow of select_values_from_door_info () stored procedure	111
Figure 6.8: Cost comparison for select_values_from_door_info stored procedure	112
Figure 6.9: Flow of update_door_info () stored procedure	113
Figure 6.10: Cost comparison for update_door_info stored procedure	114
Figure 6.11: Flow of select_object_info () stored procedure	115
Figure 6.12: Cost comparison for select_object_info stored procedure	116
Figure 6.13: Flow of update_object_info () stored procedure	117
Figure 6.14: Cost comparison for update_object_info stored procedure	119
Figure 6.15: Flow of get_routing_details_for_a_specific_day () stored procedure	120
Figure 6.16: Cost comparison for get_routing_details_for_a_specific_day stored procedure	121
Figure 6.17: Floor arrangement for first-time navigation.....	122
Figure 6.18: Floor arrangement for second-time navigation.....	122
Figure 6.19: Check route availability in BIMcache data.....	123
Figure 6.20: User navigate to the desired destination within simulator	123
Figure 6.21: Generated real-time map.....	124
Figure 6.22: Loaded movement information for scenario 1	124
Figure 6.23: Output of query execution without SQL JOIN	125

Figure 6.24: Output of query execution with SQL JOIN	126
Figure 6.25: Summary of time reduction comparisons	127
Figure 6.26: Execution time of all 8 stored procedures with and without indexing.....	128
Figure 6.27: Query execution time with and without the use of BIMcache.....	130
Figure 6.28: Total real-time map generation time comparison	13434
Figure 6.29: Linear association between number of users and execution times (without optimization).....	13939
Figure 6.30: Linear association between number of users and execution time (with optimization).....	14141
Figure 6.31: Query execution with respect to time and size.....	14343
Figure 6.32: Query execution with respect to time and size.....	145
Figure 6.33: Sample query.....	145
Figure 6.34: Comparison between the solution described in the referenced paper and AccessBIM framework.....	147
Figure 6.35: Proposed algorithm of the referenced paper [78].....	148
Figure 6.36: Query used in the reference paper [79]	149
Figure 6.37: Query used by the AccessBIM framework	149
Figure 6.38: Comparison between the frequency of execution.....	151
Figure 7.1: Comparison between query execution time and total map generation time with and without database optimization.....	156
Figure 7.2: Recommended database optimization model.....	158

Figure 7.3: Recommended real-time map generation time.....159

List of Tables

Table 2.1: Common Eye Conditions, Corresponding Impact on Vision, And Related Educational/reading Considerations [27].	100
Table 2.2: Difference between totally blind and partially sighted navigators	122
Table 3.1: Data table of registration	41
Table 3.2: Data table of Building_info	42
Table 3.3: Data table of Floor_info	42
Table 3.4: Data table of Xml_info	43
Table 3.5: Data table of Path_history_info	44
Table 3.6: Data table of Object_info	45
Table 3.7: Data table of Movement_info	47
Table 3.8: Data table of Direction_info	48
Table 3.9: Data table of Label_info	49
Table 3.10: Data table of Wall_info	50
Table 3.11: Data table of Door_info	51
Table 6.1: Query execution time with and without the use of BIMcache	129
Table 6.2: Analysis based on the total real-time map generation time	133
Table 6.3: Summary of research results	135
Table 6.4: Comparison between the number of occurrences and execution time without optimization	138

Table 6.5: Comparison between the number of occurrences and execution time with optimization.....	140
Table 6.6: Technical comparison of the two models.....	144
Table 6.7: Comparison between the solution described in the referenced paper and the AccessBIM framework	146
Table 6.8: Frequency of execution with changing number of users	150
Table 7.1: Summary of research results.....	155

List of Equations

Equation 1: Total execution time of stored procedures without indexing.....	128
Equation 2: Total execution time of stored procedures with indexing	128
Equation 3: Index efficiency factor.....	129
Equation 4: Query execution time.....	131
Equation 5: Optimization factor.....	131
Equation 6: Total time taken for map generation.....	132
Equation 7: Linear association between the number of users and execution time without optimization.....	139
Equation 8: Linear association between the number of users and execution time with optimization.....	141
Equation 9: Frequency of execution.....	149

Chapter 1 – Introduction

1.1 Problem Overview

The updated factsheet (October 2017) of the World Health Organization stated that 253 million of the world's population are estimated to be vision impaired; out of which 36 million are blind and 217 million have poor vision [1]. In addition, it has been stated that approximately 90% of the vision impaired individuals live in developing countries [2] where they require to invest money for portable navigation appliances and devices, despite poverty. Besides, a research done on the cognitive development and behavior of vision impaired individuals [3] indicated that they often face problems in socializing as although they have a desire to move along they may have fears about how to go about it. Hence, the author aims to introduce a mechanism that enhances the life-style of vision impaired individuals to facilitate their interaction with the world without difficulty.

Indoor Navigation is a dynamic and evolving research area that can contribute significantly towards providing a variety of services towards individuals with and without vision impairment, thus enabling the vision impaired to have seamless experiences related to indoor navigation.

Thus, this research aims to develop a framework for capturing indoor environmental characteristics by the use of an improved schema design that uses database optimization techniques such as stored procedures, indexing, query rewriting and BIMcache. The author believes that the proposed framework would improve the quality of life for people with vision impairment through the generation of a real-time indoor map that would guide them in their navigation. Most modern indoor environments consist of complex architectural structures with varying arrangement of physical objects. Since retrieving indoor location information has been challenging for the vision impaired, it would be helpful if spatial information of doors, walls and staircases were made available.

To address the above-mentioned problem, this thesis presents an improved schema design, an Accessible Building Information Model (AccessBIM) which could be used for generating an indoor map that could instruct vision impaired individuals in navigation, by the classification

of real world objects and their locations. AccessBIM is a real-time relational database, which acts as the main component of the central system implemented to manipulate crowdsourced data such as the floor plan and architectural data along with semantic information within the built environment. The AccessBIM database stores information on the indoor arrangement of objects within buildings to facilitate the exchange and interoperability of real-time information. The database is equipped with an optimization algorithm that reduces the query execution time with the support of indexing, query re-writing, schema redesigning and a memory optimization technique introduced as “BIMcache”.

1.2 Motivation for indoor map generation

The indoor environment plays a major role in one’s daily life as a large proportion of their time is spent within buildings in fulfilling day-to-day tasks. Expansions in the fields of architecture and interior designing have led to the introduction of more attractive, work friendly and complex indoor environments, making navigation difficult even for individuals with fair vision. Therefore, special assistance should be provided for individuals with vision impairment, so they could navigate easily through indoor environments.

In the recent past, navigation within buildings were facilitated by the use of 2-D maps created through Computer-Aided Design (CAD) applications. However, the information presented in these maps are insufficient for the vision impaired as they do not contain detailed information on the physical arrangement of objects [4]. Therefore, additional details such as the location of doors and staircases should be provided to ease their navigation.

GPS, the most common positioning technology offering outdoor localization information exhibits poor indoor performance due to its weak signal strength as GPS signals are not designed to penetrate through most concrete materials. Similarly, despite the fact that several other indoor positioning techniques have been developed, most of them are either costly or utilize fixed references to determine the location of tagged devices [5]. Hence, this created a need for an effective indoor navigation system capable of tracking environmental changes in real-time. Thus, this research examines how the capturing and storage of indoor environmental

characteristics would result in generating a real-time map equivalent to the actual environment.

1.3 Motivation for database optimization

Real-time map generation is based on characteristic changes that occur in an indoor environment. In order to generate a map of an indoor environment, real-time changes of the indoor structures (structure of the rooms, partitions, obstacles and open areas) should be stored in a database. Performance is a major issue when it comes to inserting and retrieving data from relational database management systems [6]. Real-time systems utilize complex queries and require rapid access to data than usual applications, thus creating a need for an efficient query execution process. This difference in how real-time database systems handle transactions is generally referred as the overload management problem. Furthermore, there are typical performance limitations of databases such as poor use of indexes and poor database configurations. Poor indexing may be caused due to the excessive and inappropriate use of indexes which may lead to higher consumption of hardware resources such as processors and disks [7].

Although certain limitations can be mitigated through the use of query optimization and indexing, some performance problems have more complicated causes. Hence, there is a necessity to investigate the source of the problem, which lies in how the database is structured and how it is being used by its client application for what purpose. Further, data access patterns and transactional boundaries should be analyzed in order to identify how they affect data models, and the inherent scalability of the solution to get to the root of the problem. This project aims to provide a comprehensive database optimization model to minimize problems related to handling data in real-time.

1.4 Research Gap

Scientists and engineers have developed laser canes, sonic mobility devices, handheld mobility devices and GPS devices to assist vision-impaired mobility [8], which has been discussed in detail in section 2.2.2. These devices assist the vision impaired by identifying obstacles that are nearby although they are incapable of generating a map to aid navigation.

Due to the rapid growth of mobile phone users across the world, statistics [9, 10, 11] indicate that vision impaired individuals are able to handle mobile devices equally as well as other individuals. Certain mobile devices utilize sensors to capture different parameters such as acceleration, velocity, temperature, pressure and humidity, which could be used to generate a real-time map that would facilitate vision impaired indoor navigation.

The most common way of tracking objects in an environment is the use of a Global Positioning System (GPS) [12]. Devices that use GPS signals, provide accurate directions when navigating in an outdoor environment. However, poor signal quality makes it difficult to capture characteristics within indoor environments. Therefore, the use of GPS tracking devices for indoor navigation is not feasible.

Inertial Measurement Unit (IMU) [13] based data collection is a successful method of capturing environmental characteristics when GPS, Wi-Fi or Bluetooth signals are unavailable. The accelerometer and compass of a smartphone enable the capturing of linear acceleration, duration and the direction of a user's movement [14]. However, a major drawback of having only IMU data for map generation is its inability to identify the user and information of the building around the navigator.

The limitations of GPS and IMU creates a need for an effective mechanism capable of capturing indoor characteristics in real-time. However, acquiring information alone is not sufficient, it needs to be managed as well. Hence, the author decided to research on a model that is capable of capturing indoor characteristics and is updated in real-time. The author believes that the framework would facilitate indoor navigation for the vision impaired.

1.5 Problem Statement

In the recent past, navigation within buildings were facilitated by the use of 2-D maps. However, the information presented in these maps are insufficient for the vision impaired as they do not contain detailed information on the physical arrangement of objects. Therefore, additional details such as the location of doors and staircases should be provided to ease navigation.

Furthermore, indoor environments are changing constantly as they undergo frequent object movements and partitioning. Therefore, collecting data in real time is essential in maintaining the accuracy of indoor maps. According to past literature and other research contributions [15, 16], it is important to gather real-time data to a centralized location.

Crowdsourcing is the practice of engaging a crowd to obtain required services, ideas, or content online [17]. The availability of many mobile devices with a variety of sensors in a given environment enable an individual to converge the data and filter them according to the nature and requirement of the scenario. Further, as an added advantage for the proposed research contribution, the crowdsourced data can be used to create a framework to produce a real-time indoor map.

The information obtained from crowdsourcing can be used to provide guidance on navigation within a building. Digital storage could play a major role in rendering and storing digital information in an optimized manner. Due to the complex structure of indoor spatial environments, navigation has become difficult even for sighted individuals. Therefore, identifying and tracking the changes of a location and storing them in a database has become an integral part of map generation.

The crowdsourced data needs to be stored, retrieved, updated and removed. In order to obtain the maximum use of the crowdsourced data, it needs to be managed efficiently. Therefore, Database optimization and fine-tuning methods are used to ensure the accuracy and efficiency of the data operation process.

1.6 Contribution and Significance

The findings of this research contribute to the welfare of the vision impaired. Articles, web site contents and journals based on surveys [9, 10, 11] indicate that most individuals of the modern society utilize mobile devices to make their lives easier. Statistically, 4.43 billion of the world's population are mobile phone users [18] which include both sighted and vision impaired users. Most of the mobile devices include inbuilt cameras that have the ability to capture and process images which could be used to extract important characteristics within an indoor environment.

The volume and variety of indoor data is increasing at a significant rate. The raw data collected from the environment can be stored as meaningful relational data inside the database schema, which can be implemented and tested accurately. The schema itself has the capability of generating a real-time map after filtering the required data.

The significant characteristics of the research project are listed as follows:

- Collect, store and filter crowdsourced data while arranging them in a relational schema, in order to feed them to the centralized system.
- Develop “Accessible Building Information Model (AccessBIM)” which is a relational database that acts as the main component of the central system for real-time map generation. AccessBIM database is used to store indoor building features to facilitate real-time indoor navigation for the vision impaired.
- Optimized databases have been used to generate real-time maps whilst the capability of the schema design and its applicability for real-time indoor navigation was evaluated using a simulation. This thesis presents a database optimization algorithm for AccessBIM that reduces the query execution time with the support of indexing, query re-writing, schema redesigning and “BIMcache”. Reduced query execution time is vital for exchanging information in real-time so that a map of unfamiliar structural environments could be generated to facilitate indoor navigation for the vision impaired. Database optimization leads to faster processing of the query, lesser stress on the database, less consumption of memory and efficient use of the database engine.

Figure 1.1 illustrates the component diagram of the research project. The diagram identifies suppliers, inputs, processes and outputs as well as the customers of the project in order to clearly signify the research contribution. Sighted individuals, as well as vision impaired individuals who intend to support the real-time map generation process, are identified as suppliers. Suppliers collect and provide inputs such as video streams, IMU data and localization information to the centralized system. AccessBIM model stores the data and processes them in an optimized manner while providing efficient data retrieval. The output of the process is a real-

time map that contains the most recent environmental characteristics of a specific indoor environment. Vision impaired individuals are the customers who gain the benefit from the generated map.

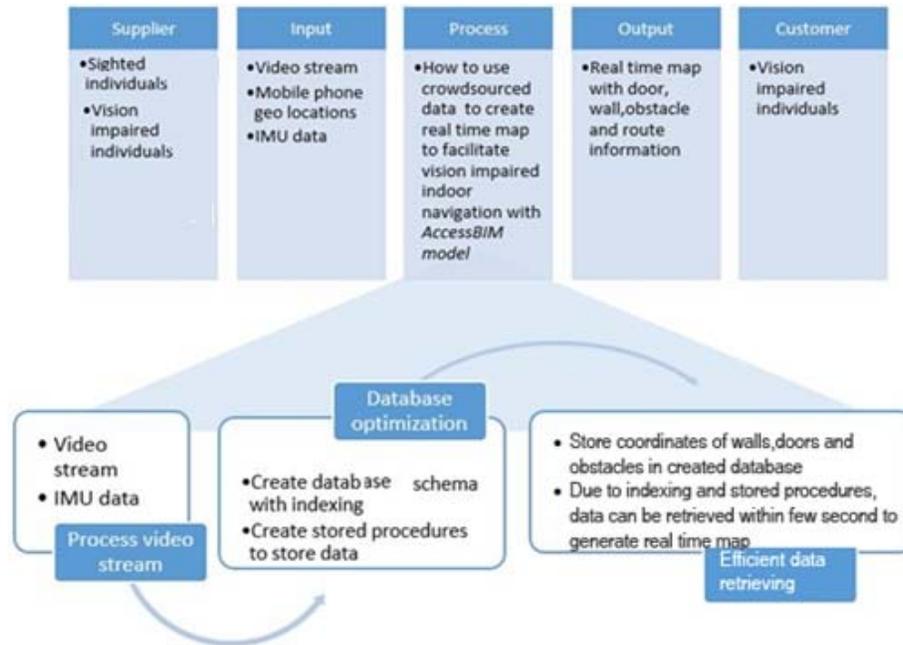


Figure 1.1: Research component diagram for AccessBIM model

1.7 Assumptions

The AccessBIM framework developed for the purpose of indoor navigation is based on image processing and IMU data [13] collected via navigators' mobile devices while they walk within the buildings. The use of IMU data with the support of GAIT analysis [19] and image processing [20] for map generation were conducted by other researchers of the same research group. This research assumes that the parameters obtained via GAIT analysis and image processing can be used to generate the AccessBIM database for which the data should be connected to the AccessBIM database via an API through the internet.

Here the data from IMU and video streams are collected and processed prior to being received by the AccessBIM. Therefore, the simulator used for the map generation process is built

according to the proposed outcomes of GAIT analysis and IMU based research work carried out by other members of the group.

The author assumes that whenever the AccessBIM database collects and stores spatial data of the indoor environment it is capable of generating an indoor map in real-time that is equivalent to the actual environment with the aid of database optimization.

1.8 Thesis Outline

This thesis is organized as follows; Chapter one describes the importance and motivation behind implementing an indoor navigation system for vision impaired individuals while chapter two describes the background of problems related with existing solutions and their drawbacks. Test framework and the stages of implementation are described in chapter three. Algorithmic implementation to obtain intended results are described in chapter four while the results of the test cases implemented in the simulation environment are discussed in chapter five. The evaluation of the AccessBIM framework together with a discussion on results is presented in chapter six. The last chapter is dedicated to the findings of the research and future work.

Chapter 2 – Background and Literature Review

2.1 Chapter Overview

This chapter discusses and compares existing solutions and methods to overcome difficulties in indoor navigation. Section 2.2 discusses the social and technical aspects of vision impaired mobility. Section 2.3 recognises and explores existing solutions for indoor navigation and map generation. As mentioned previously, the database plays an important role in the map generation process, hence section 2.4 describes the existing real-time databases and optimization mechanisms. Section 2.5 discusses the object relational implementation while section 2.6 summarizes the advantages and disadvantages of real-time map generation, and database optimization models. Section 2.7 describes the author's contribution towards the research project while section 2.8 concludes the chapter.

2.2 Vision Impaired Mobility

2.2.1 Social Aspect of Vision Impaired Mobility

Most of the vision impaired individuals find it difficult to navigate through unfamiliar indoor and outdoor environments [21]. The results of the survey done by Reginald G. Golledge et.al. [22] indicated that knowing the current location, identifying nearby features, and obtaining guidance on turns and stops are the most difficult information to obtain as a vision impaired individual. Difficulties in navigation may cause stress and anxiety [23], which may lead the vision impaired individuals to avoid leaving home or visiting complex spatial environments such as shopping malls, without assistance [24].

In the recent past, navigation within buildings were facilitated by the use of 2-D maps created through Computer-Aided Design (CAD) applications. However, the information presented in these maps are insufficient for the vision impaired as they do not contain detailed information on the physical arrangement of objects [4]. Therefore, details such as the location of doors, staircases and slopes, etc should be provided to ease their navigation.

It is a generally accepted fact that human beings including the vision impaired individuals desire to travel from one place to another. While some may wish to travel independently,

certain vision impaired individuals may wish to be brought everywhere. Accordingly, the vision impaired would experience moving in an environment, which has various objects and obstructions [25]. According to the Society for Accessible Travel and Hospitality, a vision impaired individual needs exceptional facilities such as special air, rail, bus and hotel transportation as they find it difficult to navigate without a cane or a guide dog [26].

Some individuals might suffer from total blindness while others might be affected later in life due to a health condition such as diabetes or an accident. Either way, all parties require different kinds of support to navigate as they have different views of the surrounding environment. There are numerous ways available to support the mobility of a vision impaired individual and the simplest device known is the long white cane that had helped millions of vision impaired personnel to find their way.

However, the manual white cane has major drawbacks such as its inability to identify the distance between an obstacle and the navigator which was later mitigated by the introduction of the smart cane [28]. Since the white cane is one of the first successful physical devices implemented to aid vision impaired individuals in their day to day navigational process, it has also become the starting point of the intervention of electronic based ICT technologies to achieve the same purpose. Thus, the white cane inspired the assistive technology research community to invent electronic devices that would aid vision impaired individuals in an efficient manner.

Since it is important to individually identify a vision impaired navigator, based on the categories of vision impairment, table 2.1 describes the common eye diseases and their effect on vision and education [27].

Table 2.1: Common Eye Conditions, Corresponding Impact on Vision, And Related Educational/reading Considerations [27].

Eye condition	Effect on vision	Educational/reading consideration
Achromatopsia	• Limited or no colour vision	• Support for eccentric viewing

	<ul style="list-style-type: none"> • Colours may be seen as shades of grey • Loss of detail 	<ul style="list-style-type: none"> • Use high contrast materials • Reduced or diffused lighting
Albinism	<ul style="list-style-type: none"> • Decreased acuity • Photophobia • Increased sensitivity to glare • High refractive error 	<ul style="list-style-type: none"> • Magnification (e.g., hand-held magnifier, electronic magnifier, screen enlargement software, telescope, etc.) • Close viewing • High contrast materials • Lighting from behind
Cataracts	<ul style="list-style-type: none"> • Reduced visual acuity • Blurred vision • Reduced colour discrimination • Photophobia 	<ul style="list-style-type: none"> • Support of any prescribed lenses • Magnification (e.g., hand-held magnifier, electronic magnifier, screen enlargement software, telescope, etc.) • Enlarged printed materials • Close viewing
Diabetic Retinopathy	<ul style="list-style-type: none"> • Increased sensitivity to glare • Lack of accommodation • Floating obstructions in the vitreous 	<ul style="list-style-type: none"> • Adequate high-quality lighting (e.g., lamps with rheostats and adjustable arms) • High contrast materials • Magnification (e.g., hand-held magnifier, electronic magnifier, screen enlargement software, telescope, etc.) • Training in the use of auditory materials due to loss of vision and tactile sensitivity

		<ul style="list-style-type: none"> • Training in the use of speech recognition input software
Glaucoma	<ul style="list-style-type: none"> • Fluctuating visual functioning • Field loss • Poor night vision • Photophobia • Difficulty in reading • Difficulty in seeing large objects presented at a close range 	<ul style="list-style-type: none"> • Support use of sunglasses, visors, or hats in bright sunlight and bright lighting indoors • Allow time for adjustment to lighting changes • Reduced glare • Adequate lighting (e.g., lamps with rheostats and adjustable arms) • High contrast materials

Table 2.2 gives an overview of some common issues faced by students who suffer from total blindness and partial blindness, based on an interview conducted with the Ceylon Employees Federation for which the questions are stated in appendix E. The aim is to understand the perspectives of vision impaired individuals with different impairment conditions as to how they face difficulties in indoor navigation.

Table 2.2: Difference between totally blind and partially sighted navigators

Totally Blind	Partially Sighted
Unlikely to use a printed map without some adoption	Can manage navigation with the support of a printed map to a certain extent
Likely to have particular mobility difficulties	Able to use low-vision aids in unknown environments
Unable to identify obstacles without the support of a cane or guide dogs	Can identify objects and avoid or accept them (able to see objects with bright colours or as shadows. Sometimes able to

	read the name board/road signs with some effort)
Depend on listening to instructions rather than observation	Do not depend on listening to instructions but is able to get a slight view of the environment to get an idea on how things work.

2.2.2 Technical Aspect of Vision Impaired Mobility

There are numerous ways available to support the mobility of a vision impaired individual. The simplest device is the long white cane that helped millions of vision impaired personnel to find their way. However, the manual white cane has major drawbacks such as its inability to identify the distance between an obstacle and the navigator later mitigated by the introduction of the smart cane [28]. Thus, the white cane inspired the assistive technology research community to invent electronic devices that would aid vision impaired individuals in an efficient manner. The main objective of modern assistive devices is to enhance the navigation assistance provided to vision impaired individuals [29].

One of the most popular electronic assistive equipment for the vision impaired is the laser cane [8]. It uses invisible laser beams to detect obstacles, drop-offs or other similar risks in the surroundings and produces a specific audio signal as an output to inform the user about the distance to the obstacle or the height of the drop-off. The laser cane has three different audio signals to indicate specific distance within a range of 12 feet. Furthermore, it provides safe navigation facilities to deaf-blind navigators by generating a vibration in a part of the cane's handle when there is an obstacle in front of the user. However, the laser cane is known to have its own drawbacks. It is relatively expensive compared to other indoor navigation facilitators whilst its use in social gatherings and public transportation is difficult.

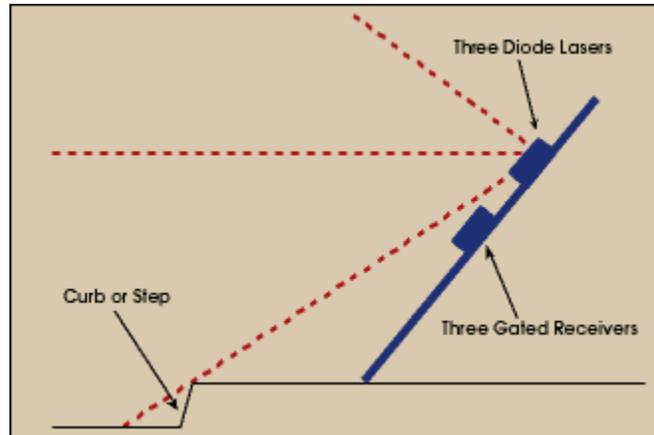


Figure 2.1: Laser Cane (Taken from: https://www.photonics.com/a16107/Lasers_Assist_the_Blind [30])

Similar to the laser cane, the smart cane is also a portable device equipped with a sensor system that consists of a vibrator, water detector and a buzzer to guide vision impaired individuals in their navigation [31]. The smart cane provides audio guidance to the speaker based on the information obtained from the environment. This device is also beneficial for individuals with aural impairment as it consists of a special vibrator glove that provides a specific vibration for each finger with each one having a different meaning. The smart cane is portable and can be bent easily which makes it convenient for use. Furthermore, it is low in cost compared to a laser cane. However, the smart cane does not detect water unless it is 0.5 cm or deep which stands out as a major drawback of the smart cane [31].



Figure 2.2: Sonic pathfinder (Taken from: <https://abledata.acl.gov/product/sonic-pathfinder> [32])

Sonic path finder shown in figure 2.2 is another well-known electrical device which will be mounted on the head of a user [8]. It uses an ultrasonic sensor to identify obstacles that are located in front of the user's path. The device uses eight musical tones to indicate the significant distance to different objects. The user can hear tones via the device's headpiece. The sonic mobility device is appropriate for outdoor use. However, it may not be used in places with extremely loud noise as the user may not be able to hear the changes in musical tones. Thus, this device is recommended to be used with a cane or a guide dog [8].



Figure 2.3: Handheld mobility device (Taken from: http://www.gdp-research.com.au/minig_1.htm [33])

Similarly, the handheld mobility device discussed in [8] is a small device that supports the navigator to determine obstacles around him with the aid of variations in vibration. Figure 2.3 illustrates a sample handheld mobility device. It produces the level of vibration depending on the distance of the object. For instance, it produces a weaker vibration for a relatively distant object and stronger vibration to a closer one. However, the handheld mobility device is recommended for use with the help of a white cane or a guide dog.



Figure 2.4: Sample GPS tracker (Taken from: <http://www.assistiveit.co.uk/VI-Products/Portable-Devices/Trekker-Breeze-GPS> [34])

Global positioning system (GPS) is the common way of identifying one's location. GPS devices as shown in figure 2.4 supports vision impaired individuals with their mobility. They use portable GPS devices to decide and verify the correct route while walking or riding a vehicle. The device includes a screen reader to hear the navigation information or braille display to read the navigation information.

All the above-mentioned devices support vision impaired individuals to travel independently, which directly supports their morale and self-improvement. Laser canes, sonic mobility devices and handheld mobility devices are recommended for the purpose of detecting obstacles while the GPS tracker is used to obtain the user's approximate locations. However, these devices are incapable of generating an indoor map due to poor localization. In addition to the above-mentioned devices, there has been research conducted by scholars to facilitate vision impaired mobility. Some of the well-known methods include Visible Light Communication (VLC), wireless technologies such as Wi-Fi positioning systems, Bluetooth low energy-based "iBeacon", Radio Frequency Identification (RFID) and image processing [35, 36, 37].

Vision impaired mobility skills may be augmented by developing an appropriate real-time database, optimizing and fine tuning the database and by generating a detailed map to guide the vision impaired individuals in an indoor environment.

Orly Lahav [38], has examined the past 15 years of research and development (R&D) on the role of virtual environments (VEs) as an Orientation and Mobility (O&M) aid [38] to enhance skills and train people who are vision impaired. The paper highlights weaknesses and strengths of using VE systems as O&M aids for people who are vision impaired. Among the identified strengths are, its design, and the availability of a flexible and adaptive learning or rehabilitation program for each client depending on their needs, hence allowing vision impaired individuals to participate in O&M activities without fear. However, certain rehabilitation centres and schools are unable to afford virtual reality mobility aids due to their expensive nature. Moreover, the difficulty in managing these virtual reality aids due to their immense size makes it impossible to be used outside a lab. However, the results of this study have the potential to influence future R&D in this field.

Since majority of the virtual environments are difficult to implement and use outside a lab, Yong Wang et al. has introduced a mobile robot to overcome the problem. Yong Wang Weidong and Chen Jingchuan Wang's paper on "Map-based localization for mobile robots in high-occluded and dynamic environments" [40] proposes a localizability-based particle infiltering localization algorithm for mobile robots to maintain its accuracy in the high-occluded and dynamic environments with moving objects. They have implemented the proposed algorithm in a campus cafeteria and a metro station using an intelligent wheelchair to evaluate the capability of the application in real-time.

The proposed algorithm is to be tested in a social welfare home in future to aid elderly and the differently abled in their daily lives. The experimental results indicate that it is necessary to consider the influence of dynamic obstacles and the previous map information during localization. The paper also concludes that the algorithm is a real-time algorithm that can maintain an accurate and robust position in the high-occluded and dynamic environment. Even though the designed wheelchair supports vision impaired or differently abled individuals to navigate without other's aid; it is a high-cost device which many disabled individuals find it challenging to afford.

Having a separate device to assist mobility is not feasible for majority of the vision impaired individuals as it incurs an additional cost to their daily expenses. Therefore, Jesus Victor et al. presented a paper on "Indoor navigation with smart phone IMU for the vision impaired in university buildings" [14]. It discusses an indoor navigation solution for the vision impaired in situations where GPS, Bluetooth or Wi-Fi signals are unavailable. The solution uses IMU, compass and barometer of the smartphone to facilitate navigation.

The paper further demonstrated how IMU sensors are used along with employing a spoken language to ease mobility (e.g. next corridor to the left; at the end of the stairs turn right, turn left, etc.) As a result, it was possible to guide the visual impaired, few hundred of meters by only using sensors of a smartphone under conditions such as the vision impaired individual strictly adhering to the guidelines provided. The major disadvantage of this research is its inability to track changes in real-time which causes the system to generate inaccurate results.

2.3 Indoor Navigation and Map Generation

2.3.1 Algorithms and Frameworks

Research conducted on converting a printed or drawn map into machine-readable format provides immense support to identify what elements should be used for real-time map generation. Ashley Beamer [41], attempts to understand issues pertaining to the classification of maps in archives and libraries. An investigation on metadata formats, such as Machine Readable Cataloguing (MARC21), Encoded Archival Description (EAD) and Dublin Core with Resource Description Framework (RDF), demonstrates how the specific map data can be stored. The practical repercussions of this work indicates the requirement for map-retrieval systems that have the capability to store metadata formats essential for retrieval. Future map catalogers should secure appropriate systems for retrieval while specifically including geographical location information such as numerical co-ordinates.

Tanaka et al. paper on “Enhanced view based navigation by mobile robots using front and rear vision sensors” [42], also provides insight into current issues in file formats used for storing map data. It also investigates the existing map-friendly systems used by libraries and archives since storage of real-time map data in metadata format optimizes search capacities. The high cost involved in using mobile robots for data collection makes it inappropriate to be used in the current study.

Chua Ching et al. [43], in their paper on “Mobile Indoor Positioning Using Wi-Fi Localization and Image Processing”, broadly discussed a two-phase framework that utilized two algorithms parallel to compensate for the other’s weakness. The algorithms used in this study used Wi-Fi Localization and image processing techniques where Localization was obtained via Wi-Fi using routers to determine the precise location of the user. Furthermore, image processing was applied to improve the accuracy of the anticipated location. Techniques such as image masking and low-resolution imagery were also integrated to improve speed without jeopardising accuracy. The tests have shown that the framework had better speed and accuracy in comparison to using these algorithms individually, and it surpassed the accuracy of a number of current indoor positioning systems. Further analysis also allowed determining the limitations of the framework, and suggestions were raised for additional refinement.

Limitations of the research include being able to collect only a few points in an area difficult to reach which may lead to poor localization.

2.3.2 Device Based Solutions

P. Benavidez, M. Muppidi et al.'s [44] paper on “Cloud-Based Real-time Robotic Visual SLAM”, presents a system and algorithm to reduce computational time and storage requirements for feature identification and matching components of Visual Simultaneous Localization and Mapping (VSLAM) by outsourcing the processing to a cloud that consists of a cluster of compute nodes. The presented solution utilizes a camera-mounted robot to take pictures of the immediate surroundings periodically in order to extract key features for map generation. By comparing features of prior images taken of the same environment by the robot, the location can be determined. The novel approach was compared to the prior one where only the local resources of the robot were used. The author identified that the increase in throughput was made possible with the new processing architecture. The major strengths of the system are, its ability to utilize computationally expensive algorithms without directly affecting robots onboard and the capability to compute functionalities and use captured images to generate an accurate map. Higher accuracy can be achieved by placing a larger number of robots inside the building. However, deploying a larger number of robots for data collection is infeasible as it incurs a high cost.

The paper by J. Tang et al. on “Fast Fingerprint Database Maintenance for Indoor Positioning Based on UGV SLAM” [16] presented a solution that uses an Unmanned Ground Vehicle (UGV) platform called NAVIS which is capable of rapidly updating the fingerprint database using indoor fingerprint information collected by employing several Signals of Opportunity (SOP) sensors. Furthermore, the magnetic field intensity was measured using a digital compass and a light sensor was used for measuring the intensity of light. A smart phone was used to collect the access point number and the Received Signal Strength Indexes (RSSI) of the pre-installed Wi-Fi network. An indoor map was generated using the NAVIS platform where the SOP fingerprint database was interpolated and updated in real-time. The efficiency and effectiveness of the proposed solution was evaluated using field tests for which the results indicated that the fingerprint database could be rapidly created and updated with a higher

sampling frequency (5Hz) and denser reference points. Despite J. Tang and his team were able to build an accurate map based on self-developed UGVs, the approach would be costly to implement in this research as the author aims to develop a solution that is affordable to everyone.

Chenglu Wen, et al.'s [15] paper on “An Indoor Backpack System for 2-D and 3-D Mapping of Building Interiors”, illustrates an indoor backpack mobile mapping system that is mainly designed for non-Global Navigation Satellite System (GNSS)/GPS. The system provides 2-D and 3-D maps for indoor environments by introducing 6-Degree of Freedom (DOF) localization. In the experiments, they discovered that the proposed Extended Kalman Filter (EKF) based method fusing 2-D laser scanning and IMU data, reduces error when the system is moving. Using IMU results of the building, for 3-D point clouds provide a more accurate 3-D map. Since this is a backpack, the navigator has to wear it while navigating, which the vision impaired individuals may find it difficult to handle without others' interaction. Another major limitation of this research would be its inability to crowdsource the collected 2D and 3D data.

In the paper “Enhanced View-based Navigation for Human navigation by Mobile Robots Using Front and Rear Vision Sensors” [42] M. Tanaka et.al, proposes an enhanced view-based navigation which is robust against featureless scenes using front and rear vision sensors. The solution utilizes two vision sensors in front and the rear of a mobile robot to have the advantage of navigating back and forth with a single recording. The solution enables human navigation in an actual environment and identify path heading for lateral wall structures.

The major concern for vision impaired individuals is that assistive technology solutions for them should be simple and easy to use as they may find it difficult to handle mobile robots or bulky backpacks.

2.3.3 Mobile Application Based Solutions

The article of J. Dhruv et al. [45] on “Design and User Testing of an Affordable Cell-Phone based Indoor Navigation System for Vision impaired” presented a cell-phone based indoor navigation system that is inexpensive and easy to use. The system includes a waist-worn user

module in addition to wall-mounted infra-red sensors and a mobile app to connect with the users. The accelerometer and infra-red sensors facilitate navigation for vision impaired individuals by locating their position in the sensor-network. As the user navigates, his current position will be updated and the navigation information would be converted from text-to-speech through the mobile application.

In order to check the accuracy, the system was implemented on several floors of a campus building and user-trials were conducted in a simulated scenario where majority of the users agreed that they were able to reach the desired destination successfully. However major drawbacks of the system include the time taken to generate navigation information through the app and the data collected via infra-red sensors not being reliable as the performance of the infra-red frequencies degrade with the increase in distance [46].

2.4 Real-time Databases and Optimization

2.4.1 General Concept

Database management systems facilitate the storage of data in a consistent and secure manner thus facilitating persistent data management, secondary storage and concurrency control and crash recovery [47]. These features ensure that transactions follow the ACID model. ACID stands for Atomicity, Consistency, Isolation and Durability. Atomicity refers to the “all or nothing rule” where either all or none of the information is committed while consistency refers to having valid data within the database, which if violated will return all data to its original state. Isolation occurs when multiple transactions running concurrently remain isolated from other transactions while durability specifies that the committed data will not be lost, regardless of system failure [48].

The ACID model facilitates consistent transaction processing while allowing concurrent control. As a security measure, every action performed on the database is written on a log before the action is made permanent in the database. Similarly, database systems also offer different indexing mechanisms which improves the performance of the system by reducing the amount of data that has to be read each time the database is accessed [48]. (Interested

readers are referred to [49] for general indexing techniques and [50] for index mechanisms regarding spatial and temporal data).

2.4.2 Real-time Databases

A real-time database uses real-time processing to handle workload whose state is constantly changing [50] to find out an appropriate real-time database for AccessBIM implementation.

PostgreSQL is an advanced, open-source object-relational database management system which is standards-compliant and extensible [52]. It is among the top 10 relational database management systems according to the DB-Engine rankings published at the end of the year 2016 [51]. The official PostgreSQL website states that there are numerous advantages [52] of using PostgreSQL over other real-time database systems.

PostgreSQL databases are immune to over-development as there is no licensing cost associated with the software while it also reduces the staffing cost due to lower maintenance and tuning requirements. Furthermore, PostgreSQL databases demonstrate extreme responsiveness in high volume environments and are equipped with high quality graphical user interface tools.

2.5 Object-Relational Implementation

The figure 2.5 illustrates the four-quadrant view of databases [53]. The lower-left quadrant represents applications that process simple data which do not require querying. They could survive with the underlying operating system to acquire crucial DBMS functionality of persistence. Standard word processing applications such as word perfect and frame maker could be considered as examples. Applications in the lower-right quadrant are required to process complex data but do not have the necessity of querying the data. Computer aided design packages are an example for Object-Oriented Databases (OODBMS) in this quadrant. The top-left quadrant is for Relational Databases (RDBMS) that process simple data but requires complex querying. Business applications belong to the RDBMS quadrant. The last quadrant is for applications that process complete data and have complex querying

requirements. Advanced database applications are recommended to use Object Relational Database systems due to their requirements belonging to the top-right quadrant.

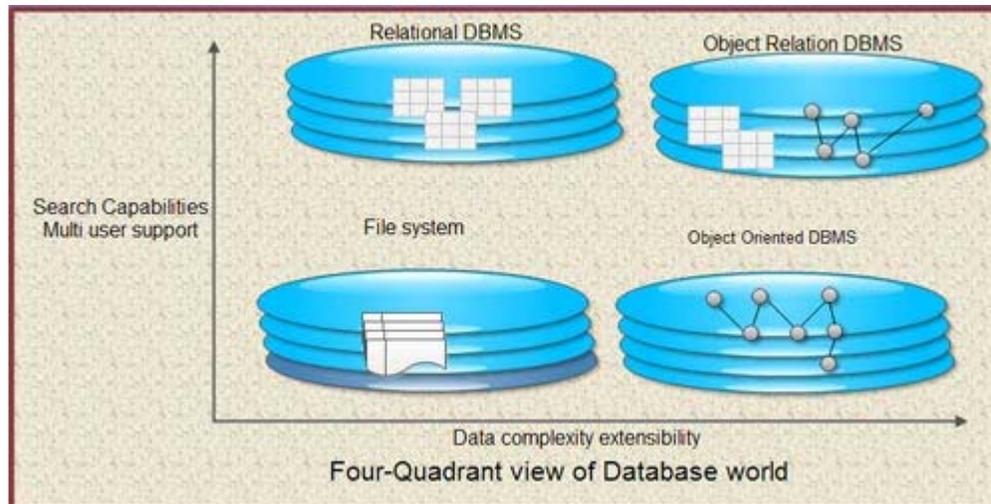


Figure 2.5: Four-quadrant view of the database world (Taken from: <http://ecomputernotes.com/database-system/adv-database/object-relational-database-systems> [53])

According to the four-quadrant view of the database world, traditional SQL-based DBMS are difficult to manage when dealing with complex data in application areas. [54]. The ORDB is a hybrid concept of the RDBMS and OODBMS. It allows inclusion of data types and methods when developing a database to enhance the ability to sort and locate a file within the database as well as facilitating efficient data filtering.

AccessBIM model for real-time map generation was implemented as an Object Relational Database (ORDB) due to several reasons. The need, to handle an enormous amount of IMU data together with the use of many stored procedures and indexes are the main reasons behind selecting an Object Relational Database despite the fact that it is complex and costly. Although databases such as NoSQL is capable of handling large volumes of data, they lack the ability to perform ACID transactions. The ACID model facilitates consistent transaction processing through atomicity, consistency, isolation and durability thus making an ORDB the most feasible database model for the AccessBIM framework.

2.6 Database Optimization and Performance Tuning

Fan Yuanyuan et al. paper on “Distributed Database System Query Optimization Algorithm Research” [50], describes a newly introduced query optimization algorithm that is based on commonly utilized optimization algorithms used in a distributed query. The designed algorithm is capable of diminishing the amount of intermediate results by a considerable amount, which results in reducing the network communication cost, leading to optimal efficiency. However, this algorithm is not applicable for the author’s approach as the AccessBIM framework does not consist of distributed queries.

P. Karthik, G. Thippa Reddy and E. Kaari Vanna's paper on “Tuning the SQL Query in order to Reduce Time Consumption” [55], presented a novel query optimization technique that is capable of improving the overall performance of the system by introducing lesser stress on the database in data transmission, thus utilizing the database engine efficiently with reduced memory requirements. However, the research does not focus on indexing strategies to further enhance time reduction. Similarly, Lin Hong, Zhuhai, Mingda Lu and Weiting Hong on “A Business Computing System Optimization Research on the Efficiency of Database Queries”, also discussed the performance evaluation, measurement, and business computing system optimization based on an experimental research on the efficiency of database queries [56] without considering the use of indexing.

Ivo Jimenez, Jeff Le Fevre et al. paper on “Benchmarking Online Index-Tuning Algorithms” [57], outlined a performance benchmark for the problem of online index tuning without the use of query optimization.

Steven W. Schlosser and Sami Iren’s paper [58] on “Database storage management with object-based storage devices” discussed how object-based storage interfaces could be used to communicate storage requirements to the storage subsystem thus making information available to low-level optimizations. Here, the entire table is stored as a single object where the relational schema is conveyed through attributes assigned to a specific object.

Zhan Li, Olga Papaemmanouil et al. paper [59] on “OptMark: A Toolkit for Benchmarking Query Optimizers” introduces OptMark, a toolkit for evaluating the quality of database

optimizers by assessing its efficiency and effectiveness. The evaluation of optimizer performance is distinguished from existing DBMS benchmarks based on the performance of its underlying DBMSs execution engine.

Nicholas Roussopoulos's paper [60] on "View Indexing in Relational Databases" describes that the design and maintenance of a helpful database framework requires a productive way for enhancing the logical access paths through repetitive usage patterns. According to [60], the secondary index selection and query optimization techniques are insufficient for optimizing a wide collection of logical access paths. However, optimal query processing cannot be achieved through individually optimizing the queries.

Based on above-mentioned papers, the author concludes to implement a conceptual framework using query optimization, indexing and few more approaches as listed in section 3.3.2. The use of query optimization results in faster information retrieval thus enabling the generation of the indoor map in real-time.

The need for an efficient query execution process is generated as the AccessBIM requires to update and retrieve data in real-time.

2.7 Contribution to the Research Project

Jayakody and Murray's paper on "The Construction of an Indoor Floor Plan Using a Smartphone for Future Usage of Blind Indoor Navigation" [61] presents an approach of creating an indoor map that can be used to facilitate vision impaired indoor navigation using smartphones. The proposed model demonstrates the ways of capturing data to generate maps and further enhancement of the algorithm towards error correction. The key aspect is to build an open, participatory and collaborative system through which users can contribute environmental information using their mobile devices to address the issues involved in the application of collaborative indoor mapping.

At present, most of the indoor positioning techniques are based on wireless LAN signals (WiFi) which is the most common infrastructure available in any type of indoor environment. However, this paper discusses the application of the "AccessBIM" framework in way finding

and data synchronization, via the generation of a real-time topological map to assist vision impaired individuals. Furthermore, the paper also discussed the four stages of collaborative indoor mapping. In the first stage, data is obtained via gait analysis and image processing using accelerometers and electronic compass sensors integrated with smartphones. Then, the footpath algorithm is utilized for navigation and pedestrian dead reckoning (PDR). PDR is the process of calculating one's current position by using a previously determined position and advancing that position based upon estimated speeds over elapsed time [61]. By tracking a user's path, PDR is capable of estimating the location of a user when the starting position, number of strides and stride length is known.

The second stage executes several sub processes such as smoothing, vertex constriction and sphere contraction. Afterwards, the relaxation algorithm is modified for error correction. The use of a relaxation algorithm facilitates the calculation of the estimated position and its proposed variation by putting every vertex to its expected position. In the third stage a spatial database was made available to store data. This database consisted of a classic navigation graph and additional vector data of probable indoor movement traces that would facilitate the generation of navigation directions. The objects in the database were classified into two main categories as “immovable obstruction” and “movable obstruction”. A movable obstruction-object could be moved here and there within the indoor environment while the immovable obstructions were static objects. Thus, the authors proposed simultaneous localization and mapping (SLAM) techniques to generate maps. Smart SLAM facilitates the automatic creation and updating of indoor maps using Wi-Fi fingerprints, Wi-Fi access points installed within buildings, as landmarks. It tracks a user's path by using an accelerometer sensor to detect the steps and an electronic compass sensor to detect directions.

Afterwards, a bounding box file is created for each map. A bounding box file is a configuration file that contains all the important parameters required to create a map. It should contain general information about the floors, coordinates of the boundaries and the storage location of the available map. In the newly proposed architecture, sensor data through image processing and gait analysis will be collected from the external environment for map generation [61]. The collected sensor data will then be directed to a specially designed API

for processing. After the accomplishment of the error correction process, the decision-making process is initiated. The final stage facilitates the convergence of objects within the AccessBIM using Java OpenStreetMap Editor to assist real time processing with multiple indoor users.

Similarly, the paper also proposed a navigation algorithm that consisted of a data acquisition and smoothing algorithm as well as a map generation and error correction algorithm. In here, the data obtained via image processing and gait analysis is directed to a specially designed API which invokes a special function called `dataSegment()`. Based on a set of predefined rules `dataSegment()` function will attempt to segment the incoming sensor data into two categories namely, temporal data and landmark based data semantics. This process is repeated until all the sensor data is segmented and stored in the corresponding tables within the database.

Afterwards, the correctly segmented data will be retrieved from the appropriate tables for the decision making. After the completion of the decision-making process SLAM will be used to localize and the indoor map will be generated using “IndoorOpenStreetMaps” (IndoorOSM).

Jayakody and Murray’s paper “Proposed Novel Schema Design for Map Generation to Assist Vision Impaired in an Indoor Navigation Environment” [62], introduces an improved schema design that could be used to generate an indoor map based on the classification of real world objects and their locations. Furthermore, the paper discussed the synthesis of input sensor data communication with the K-SOAP protocol to facilitate real-time map generation with a special cache memory incorporated in the cloud database. This paper described five major tables and their respective relationships to address the issues involved in the application of collaborative indoor mapping.

The five relations were: `Object_info` table, `Label_info` table, `Movement_info` table, `Direction_info` table and `Localization_info` table. The `Object_info` table stores the `object_id` projected to generate automatically, `object_description`, `received_timestamp` and the category of the object whether it is movable or immovable [62]. The field `timestamp` defines the time at which the data was received to the table as it would be an important parameter in determining the novelty of the data in generating a real-time map. In the meantime, the

label_info table contains label information of different places that the user visits. It consists of fields such as the label_name, label_image, description of the place and the time stamp.

The movement_info table contains data received via mobile phone's sensors. It consists of the user_id, R_value which is the user's speed towards the object and the theta_value which is the angle of the object respective to the user. The direction_info table contains the path information to a particular object relative to a given user. It consists of the image_id, i_value which is the angle of the object relative to the user and the distance. The localization_info table contains data received through the mobile sensors in terms of X, Y and Z coordinates. Thus, its fields include the location_id, X coordinate, Y coordinate and Z coordinate.

The K-SOAP protocol was used to upload sensor data and download the map data information from the cloud. Similarly, a caching technology was used to keep up the communication between the front end mobile device and backend database as well as to identify paths that are already existing within the database, thus reducing the time of retrieving data [62].

Thus, as an extension of the above-mentioned research work the author uses an improved version of the novel schema design together with four algorithms to facilitate the generation of a real-time indoor map with the application of open and participatory crowdsourced data.

2.8 Chapter Conclusion

This chapter discussed the existing solutions for vision impaired mobility, indoor navigation and map generation, real-time databases and database optimization.

The social and technical aspects of vision impaired mobility help to identify the need for a real-time indoor map that facilitates navigation for the vision impaired. Although devices such as laser canes, sonic mobility devices and handheld mobility devices are available, they are only capable of identifying obstacles and the distance to the obstacles. Hence, there is a need for a complete solution that could direct the vision impaired individuals to their desired destination. The model introduced in this thesis is able to generate an accurate map based on real-time environmental characteristic changes.

Key benefits achieved by the use of AccessBIM framework to generate an indoor map in real-time include:

- **Time saving:** Data collection via crowdsourcing and IOT reduces the time in collecting data compared to collecting data from robots.
- **Less confusion:** As the map is being updated regularly on the changes of the indoor environment, it would be simple and easy to use.
- **Increase productivity:** The real-time map will be generated much faster due to the use of database optimization techniques.

The AccessBIM database is capable of achieving the above-mentioned benefits due to faster query processing, lesser cost per query and efficient use of the database engine with less memory consumption. Indexing makes it convenient to search the required data from large data sets while query optimization facilitates the identification of the best query for retrieving data using minimal resources thus enabling high performance of the system via faster processing and lesser database cost thereby increasing the performance of the real-time map generation layer. This leads to efficient usage of the database engine with less memory consumption as the database is generated without data duplication, while indexing and query optimization enables faster data insertion, updation, deletion and retrieval.

Generation of real-time map depends on the data stored in the AccessBIM database. Hence, it is necessary to employ appropriate data optimization mechanisms to retrieve data within few seconds. Existing research and studies on database optimization are focused on query optimization as well as indexing. However, most of them focus on either query optimization or indexing. The improved database schema introduced in this research is able to generate a highly optimized database with the support of both query optimization and indexing. Apart from query optimization and indexing, the research also focuses on query rewriting and schema alterations to further enhance database optimization.

Chapter 3 – Framework, Database and Test Environment Design and Implementation

3.1 Chapter Overview

This chapter discusses the framework that leads to the database design, its implementation and finally, testing the environmental design that was used to develop the solution. Section 3.2 discusses the experimental design objectives and the research questions while section 3.3 describes the implementation phases, methodology, database design, optimization techniques and the AccessBIM framework in detail. Section 3.4 describes the real-time map generation process while section 3.5 summarizes and concludes the chapter by discussing the methodology that gives effective, procedural guidance to generate an indoor map by making use of the data stored within the AccessBIM.

3.2 Experimental Design Objective

The research questions stated below are related to establishing an efficient, optimized data-structuring framework with the support of a navigation simulator by exploring what type of tools and stored procedures are required to generate a real-time map for vision impaired individuals. The key aspect of the chapter is established as follows:

- How are the indoor structural characteristics stored, optimized and used to generate a real-time map of an indoor environment?

The focus of this research is to adopt crowdsourced data related characteristics in an indoor environment, to generate a real-time map that assists vision impaired individuals in their navigation.

3.3 Implementation Phases of the Research Work

This research work presents an experimental technique for developing a solution that gives effective, procedural guidance to construct a data structuring framework that facilitates the generation of an indoor map by making use of the data stored inside the AccessBIM database. Major phases of the research procedure are illustrated in figure 3.1.

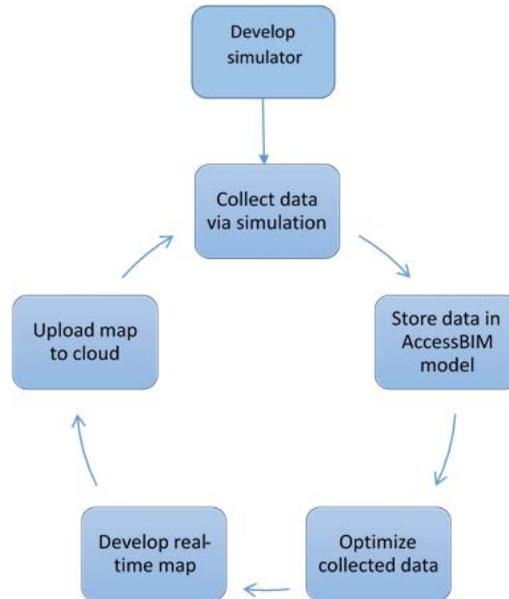


Figure 3.1: Main phases of AccessBIM model Implementation

3.3.1 Methodology of Framework Implementation

This thesis introduces the AccessBIM framework of environmental characteristics, proposed by the author to store indoor building features that facilitates real-time indoor navigation for the vision impaired. AccessBIM stands for Accessible Building Information Model that eases accessibility of the vision impaired as it aids the management of indoor spatial data in a digital format to assist vision impaired individuals to independently access unfamiliar indoor environments. The AccessBIM framework comprises of a layered architecture where each layer performs a specific role within the application. The use of layered architecture enables responsibility to be shared among the layers in addition to generalizing the work that needs to be done to satisfy a specific user request. Besides, frameworks that use layered architecture are easy to develop, test, govern, and maintain [63].

Figure.3.2 illustrates the AccessBIM framework with its five layers namely; Data Collection Layer (DCL), User Interface Layer (UIL), Function Layer (FL), Database Optimization Layer (DOL) and the Backend Layer (BL). The data collection layer collects crowdsourced data via IMU, sensors and IoT enabled devices. Next, the collected data will be sent to the Database through the API in the User Interface layer. The functional layer consists of four sub processes namely: initiating the database connection, calling for function, returning the function call and terminating the connection with the database. The queries are optimized and indexed in the Database optimization layer while the backend layer contains the schema design and stored procedures that filters only the required data to generate the map.

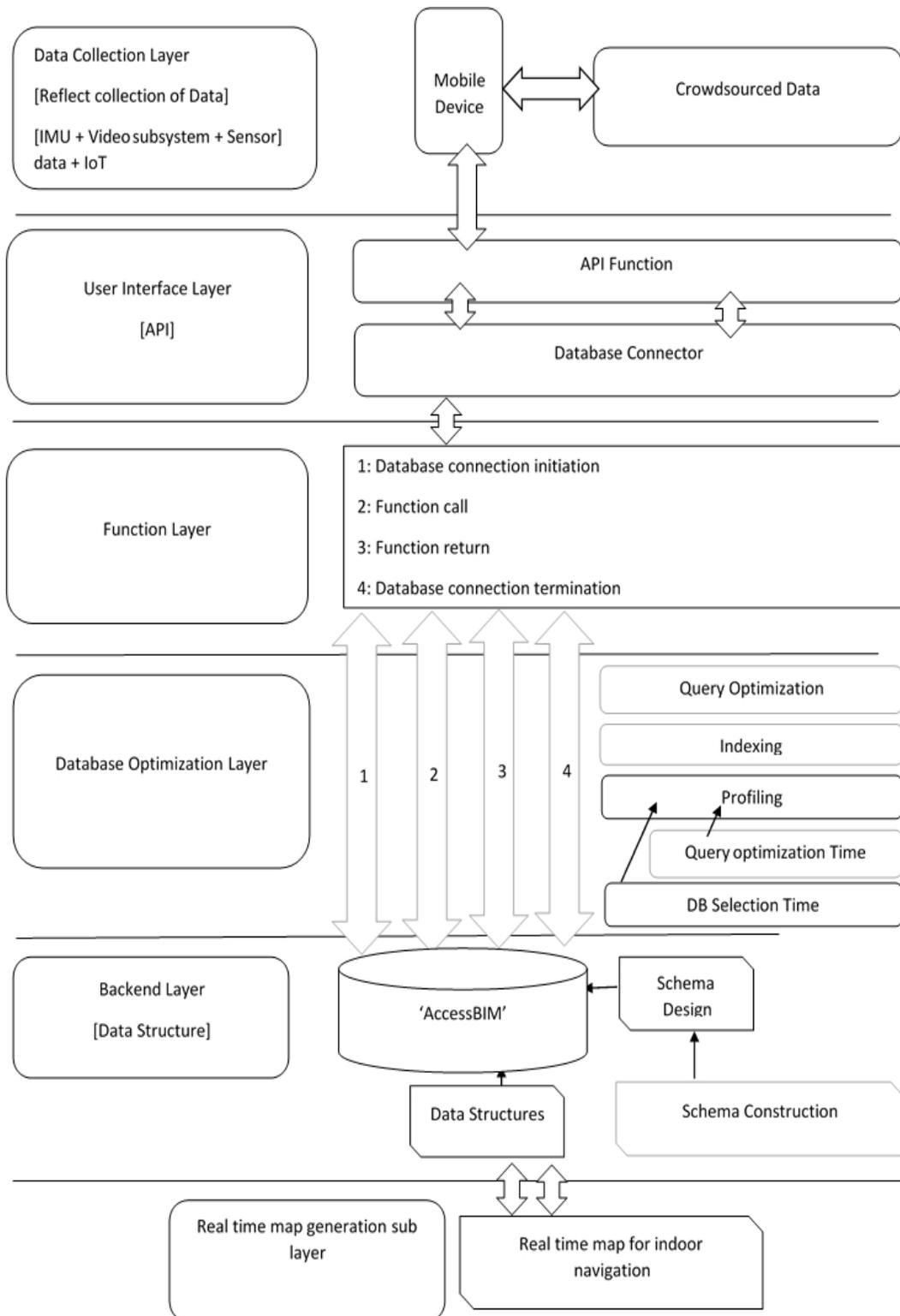


Figure 3.2: AccessBIM Framework

3.3.1.1 Data Collection Layer (DCL)

Nowadays, smartphones and mobile devices are equipped with different tools such as cameras, compasses, light sensors etc. The aim of the data collection module (DCL) is to collect crowdsourced data that are inputs to the system.

Environmental details could be collected using mobile sensor data. Accelerometers provide details on acceleration force in ms^{-2} including the force of gravity. Gyroscopes provide the rate of rotation in radians per second for each of the three physical axes. Additionally, modern buildings equipped with sensors, wireless devices and IOT enabled equipment could be used as sources of data collection.

In addition to acquiring data through sensors, wireless devices and IoT enabled equipment, the author assumes that any form of crowdsourced data collected through PGM and GAIT analysis could also be added to the AccessBIM database, as PGM and GAIT analysis were utilized to acquire environmental data by other researchers of the same research group.

Image processing data defines objects and their details using a video subsystem. N. Rajakaruna et al. [20] discussed the possibility of using an image-based object detection algorithm on an object's edges and corners rather than its distinctive characteristics such as color and texture. A Probabilistic Graphical Model (PGM) was used for extracting features and a generic geometric model was built to detect objects by coalescing edges and corners. Additionally, supplementary geometric information was also utilized to differentiate objects with similar size and shape (e.g., bookshelf, cabinet, etc.).

The collection of indoor environmental data also includes data collected through gait analysis [19]. The gait analysis system consists of a sensor unit placed on smartphones, which provides the direction and distance to an object in relation to the source of data collection, as "R-Value" and " θ Value" which is then stored in the AccessBIM database. N. Abhayasinghe et al. [19] have discussed the possibility of using the data of, a thigh mounted internal measurement unit (IMU) to detect the movement of the thigh, and the number of steps accurately.

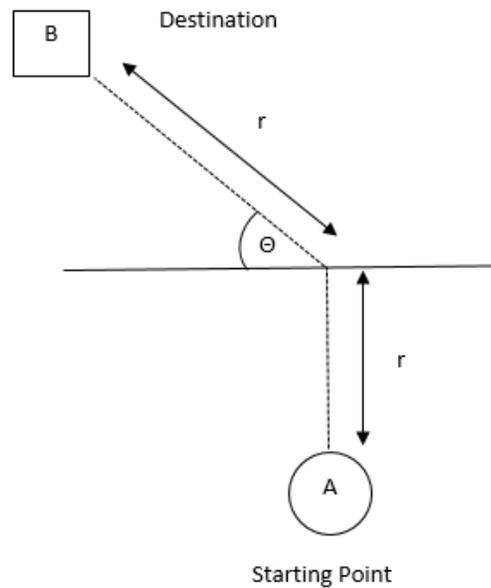


Figure 3.3: Collecting data through GAIT Analysis

Figure 3.3 illustrates how GAIT Analysis measures the “R-value” and the “ θ Value” as a user navigates within an indoor environment from starting point A to destination B. The collected data could be used to measure the walking distance, and direction with angle of movement of the vision impaired individuals which is later added to the movement_info table to be used in map generation.

The use of crowdsourcing facilitates data collection through the use of existing devices. However, the use of crowdsourcing in data collection has its own limitations such as there can be no guarantee that the results of crowdsourcing will be of sufficient quality for map generation as there could be variations in output due to limited signal strength and device diversity. Furthermore, the data collected via crowdsourcing needs to be cleared out to obtain the most relevant and recent data to build the real-time indoor map as per the user’s requirement. Besides, crowdsourcing being the practice of engaging a crowd to obtain content online [17] it would lead to the capturing of confidential environmental information that require to be treated securely.

3.3.1.2 User Interface Layer (UIL)

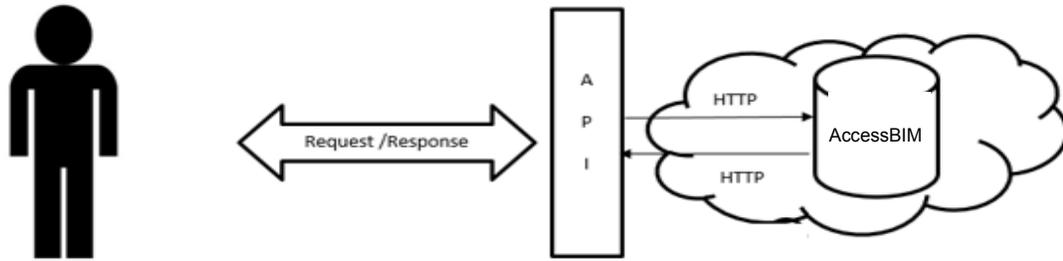


Figure 3.4: User Interface Layer

The AccessBIM database is implemented in a cloud environment. Hence, the suppliers of crowdsourced data could send data to the database via the cloud as well as retrieve the generated real-time indoor map through the Application Program Interface (API). API is the key element, which is responsible for handling and controlling the overall functionality of the system including the connectivity with the database. The API primarily consists of two components, namely DB connector and the API functions.

API includes a database connector (Open Database Connectivity (ODBC), Java Database Connectivity (JDBC)) and the API functions (insert, update and delete) to send and receive required information, to assist vision impaired individuals in an unknown environment via their mobile devices. Since the AccessBIM model is implemented within a cloud environment, it ensures better performance and availability together with greater flexibility and scalability to provide an accurate and continuous service in map generation for vision impaired individuals.

3.3.1.3 Functional Layer (FL)

The functional layer is the main connectivity module which takes the responsibility of connecting the user with the database optimization layer in order to utilize the AccessBIM database as the back end.

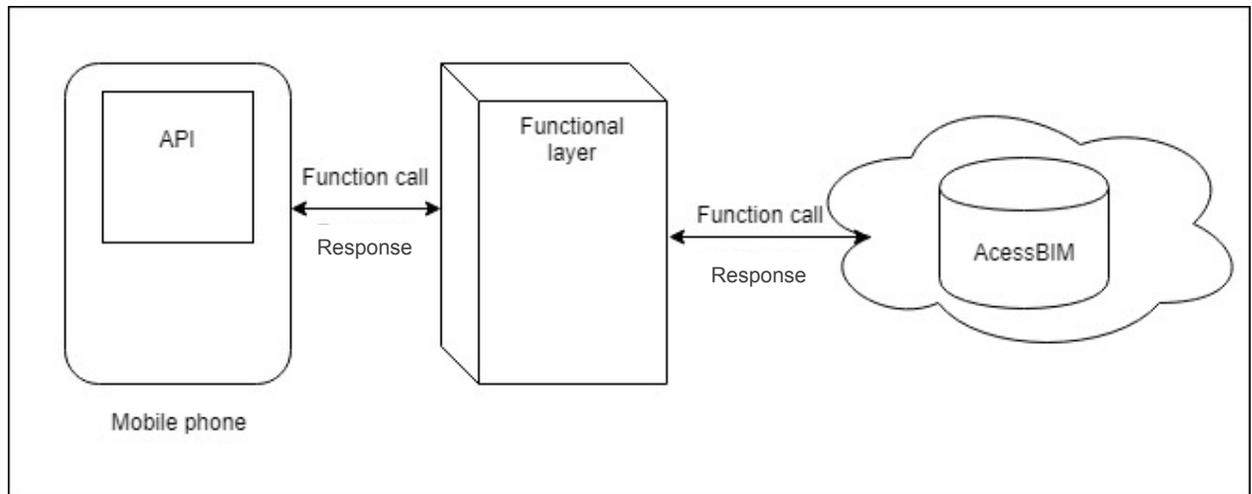


Figure 3.5: Process of functional layer

As shown in figure 3.5, the API in the mobile phone initiates the connection with the database through a function call which is received by the functional layer and the AccessBIM database respectively. The database responds through a function return thereby terminating the database connection.

3.3.1.4 Database Optimization Layer (DOL)

The insertion, updating, deletion and retrieval of data could be delayed due to the high volume of data entries exchanged between the users and the system, thus slowing down the process of map generation. Hence, the Database Optimization layer in the AccessBIM model utilizes query optimization techniques, to reduce the time of query execution. The main role of this layer is to fine tune the database through indexing, caching and profiling, to improve performance in terms of reduced, database selection time, query optimization time and faster retrieval of stored information.

3.3.1.5 Backend Layer (BL)

The backend layer is the core of this research project. The backend layer is divided into two sub-layers namely the storage sub-layer and the real-time map generation sub-layer.

a) Storage Sublayer

The storage sub-layer has the ability to store, processed and optimized data as a dataset that contains inter-related tables after normalization, anomalies and redundancies.

b) Real-time Map Generation Sublayer

The real-time map generation layer facilitates generating a map related to most recent characteristics in an indoor environment. The data collected through the API is stored inside the AccessBIM model after being successfully optimized. Optimized queries contribute in obtaining accurate data from the AccessBIM model to generate a real-time map when requested by a vision impaired individual.

3.3.2 Database Designing Approach

3.3.2.1 Database Design

The ER diagram (Figure 3.6) illustrates how the database schema organizes tables to store indoor data. At the beginning of the real-time indoor map generation research, the database design was based on object, label, direction, movement and location information of the user.

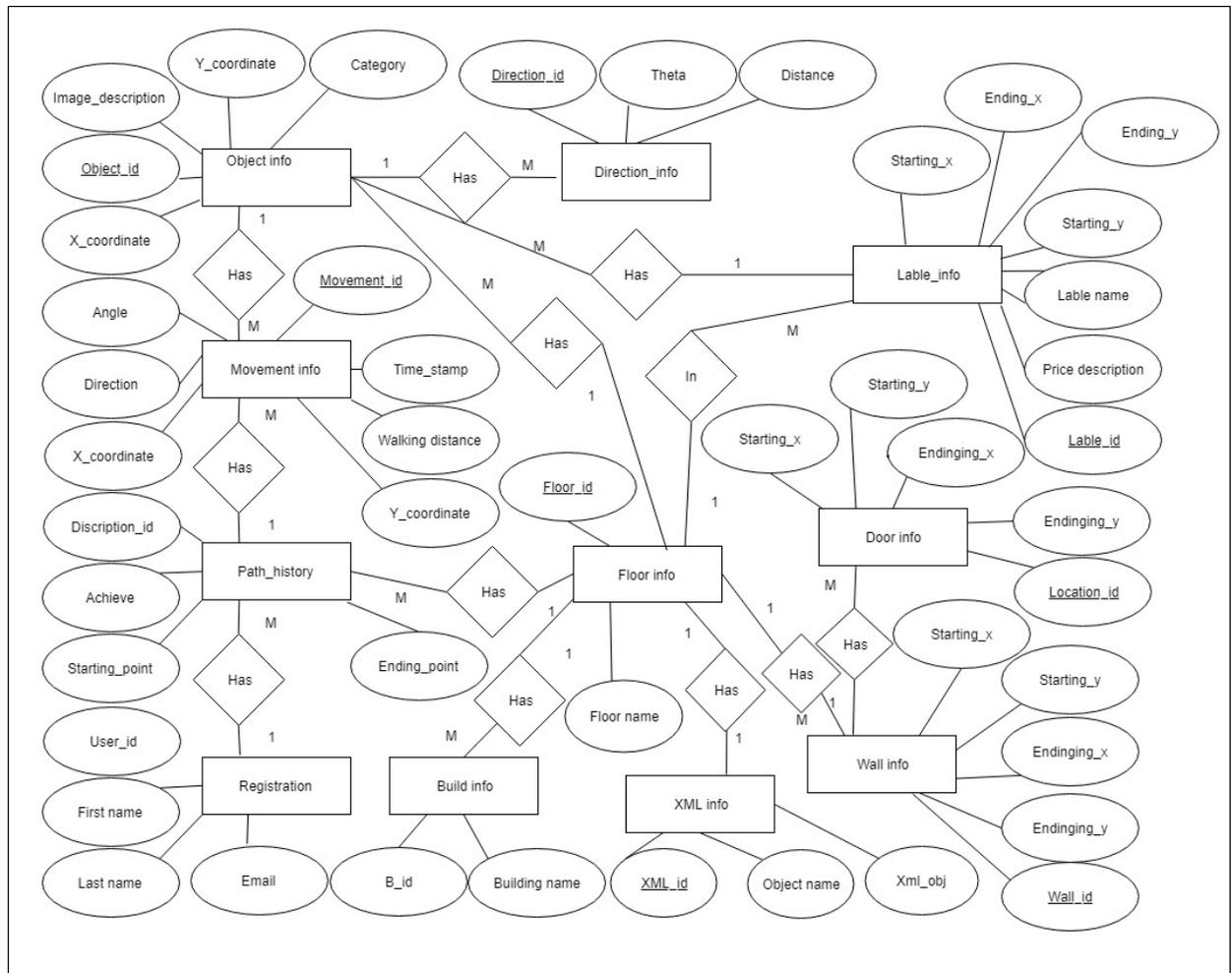


Figure 3.6: ER Diagram with Table relation

However, the discussion with the vision impaired employees of ‘The Employers’ Federation of Ceylon’ implied that there should be a way of storing floor details, building details and paths available with their starting points and end points in order to generate an accurate map. Hence, the entities and attributes to build the ER diagram were identified from an interview conducted with vision impaired individuals at “The Employers' Federation of Ceylon”. The questions used in the interview are available on Appendix E. Figure 3.7 illustrates the complete relational schema for building the AccessBIM database based on user requirements identified in the set of interviews conducted with the vision impaired individuals of EFC.

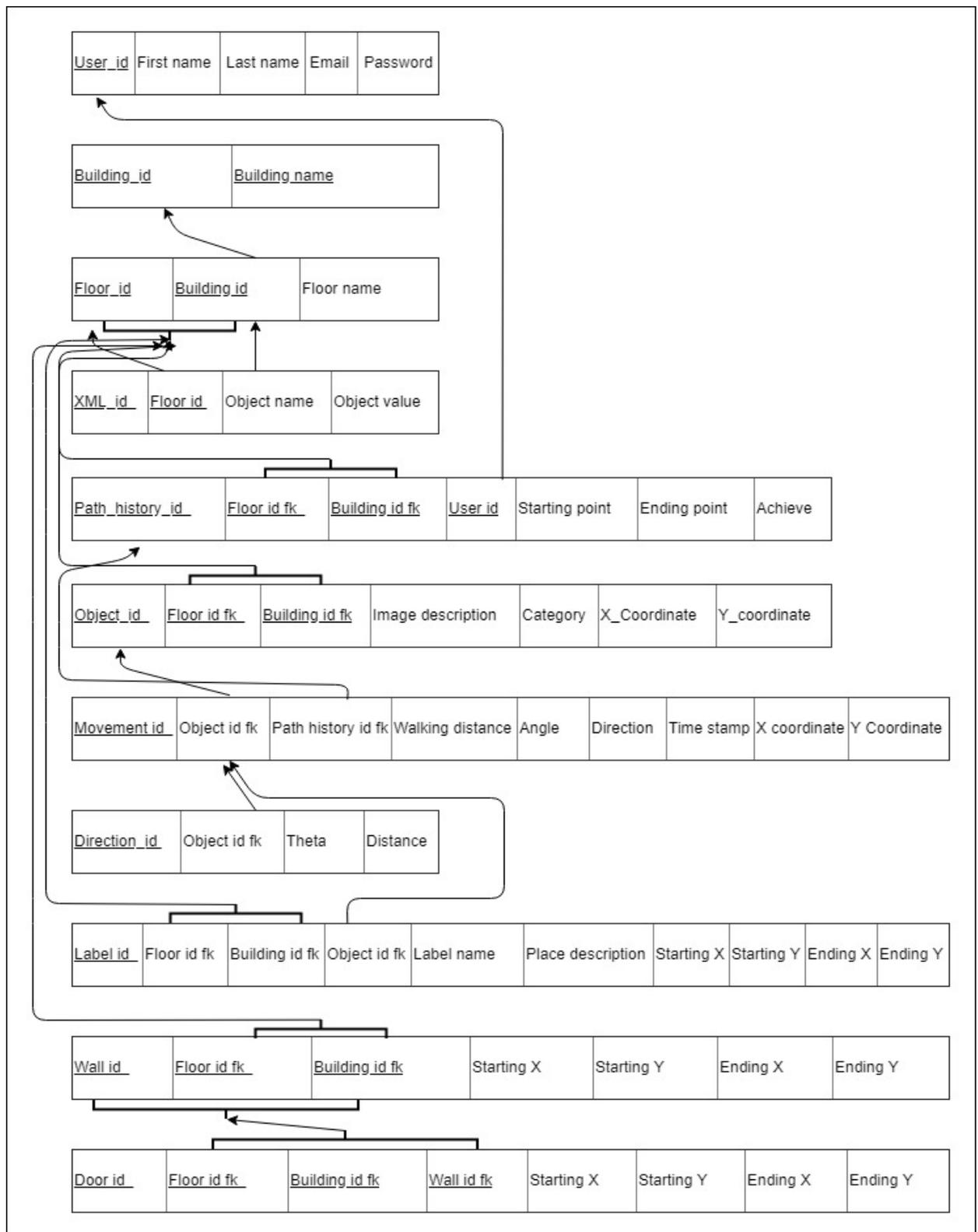


Figure 3.7: Schema Design with relations

The relational schema in figure 3.7 represents entity relationships with the primary and foreign keys of the tables in the AccessBIM database, as mentioned below.

a) Registration table

Table 3.1 describes the columns and rows of the registration table. It has ‘User_id’ as an integer type master key, which is projected to generate automatically by the schema. It is followed by user details such as First name, Last name, Email and Password. ‘User_id’ acts as a foreign key to the path_history table’ to identify each user separately when navigating.

Table 3.1: Data table of registration

User_id	First_name	Last_name	Email	Password

Column description:

User_id: Primary key. A unique entry to identify each user

First_name: String type tag to store the user’s first name

Last_name: String type tag to store the user’s last name

Email: String type tag to store the user’s email address

Password: String type tag to store the user’s password

b) Building_Info table

The ‘Building_Info table’ (Table 3.2) contains information about the building. It consist of two columns to store the ‘Building_id’, which is the primary key and the ‘Building_name’. The ‘Building_id’ also acts as a foreign key to ‘Floor_info table’ to facilitate the identification of floors uniquely. The ‘Building_name’ will be filtered accordingly, after the video streaming data of the building is processed. (i.e.: When the user goes through the door of the building, it will be recorded in the mobile device. If the entrance door contains the name of the building, it could be filtered out and added to the ‘Building_name’ column)

Table 3.2: Data table of Building_info

Building_id	Building_name

Column description:

Building_id: Primary key. A unique entry to identify each building

Building_name: String type tag to store the name of the building.

c) Floor_info table

Table 3.3 contains floor information about each floor within a building. It contains three columns to store 'Floor_id', which is the primary key, the 'Building_id' and the 'Floor_name'. 'Building_id' is a foreign key from 'Building_info table' so that buildings could be identified uniquely. 'Floor_name' will be filtered out after processing the video streaming data of the floor information, not immediately but accordingly. (i.e.: When the user enters a particular floor, it will be recorded in the mobile device of the navigator. If the entrance door contains a label of the floor name it could be filtered and added to the 'Floor_name' column). 'Floor_id' is a foreign key to the 'Xml_info table'. 'Floor_id' together with the 'Building_id' acts as a foreign key to 'Path_history_info', 'Object_info', 'Label_info' and 'Wall_info' tables to recognize building and floors.

Table 3.3: Data table of Floor_info

Floor_id	Building_id_fk	Floor_name

Column description:

Floor_id: Primary key. A unique entry to identify each floor

Building_id_fk: Foreign key, which refer to Building_info table

Building_name: String type tag to store the name of the floor.

d) Xml_info table

The 'Xml_info table' (Table 3.4) stores information such as the name and value of each floor in terms of XML (Extended Markup Language). Auto incremental 'Xml_id' is the primary key of the table. 'Object_name' stores the floor name while the 'Object_value' column stores the floor details, as XML values. After collecting and organizing floor data via IMU and video stream data, the data can be converted into XML format and stored inside the 'Xml_info table'. Storing data in XML format reduces the time of retrieving floor information from the database.

Table 3.4: Data table of Xml_info

Xml_id	Object_name	Object_value

Column description:

- Xml_id: Primary key. A unique entry to identify each floor
- Object_name: Floor name in XML format
- Object_value: Floor details in XML format.

e) Path_history_info table

Table 3.5 named as the 'path_history_info' is used to store details about the source and destination. It contains columns such as 'Path_history_id', 'Floor_id_fk', 'Building_id_fk', 'User_id', 'Starting_point' and 'Ending_point'. 'Path_history_id' is the primary key that indicates each distinct path within the given source and destination. In addition, it acts as a foreign key to the 'Movement_info table' to track user's navigation in a specific path. 'Floor_id_fk' and 'Building_id_fk' are foreign keys from 'Floor_info table' while the 'Building_info' table figures out the location of the path. 'User_id' is a foreign key from the Registration table to identify the specific path of each user in navigation. 'Starting_point' stores details of the source while the 'Ending_point' stores details of the destination. Details on the Starting point is captured via localization information as vision impaired individuals

start their journey while the ending point column is auto-filled through the processing of real time video stream data as the navigator ends his journey. The ‘Achieve’ column tracks whether the user successfully reaches the destination or not by using Boolean value.

Table 3.5: Data table of Path_history_info

Path_history_id	Floor_id_fk	Building_id_fk	User_id	Starting_point	Ending_point	Achieve

Column description:

Path_history_id: Primary key. A unique entry to identify each path

Floor_id_fk: Foreign key, which refers to Floor_info table

Building_id_fk: Foreign key, which refers to Building_info table

User_id: Foreign key, which refers to Registration table

Starting_point: Localize information when vision impaired individual starts the journey after processing video stream data.

Ending_point: Localize information when vision impaired individual ends the journey after processing video stream data.

Achieve: Uses Boolean value to capture whether the user has reached the destination successfully.

f) Object_info table

The ‘Object_info table’ (Table 3.6) is used to record details about objects that are identified on a certain floor. It contains columns such as ‘Object_id’, ‘Floor_id_fk’, ‘Building_id_fk’, ‘image_description’, ‘Category’, ‘X_coordinate’, and ‘Y_coordinate’. ‘Object_id’ is a unique key, which is generated automatically when a user finds a new object/obstacle on the floor while navigating. Objects could be identified after processing the video stream data provided by a vision impaired individual. ‘Object_id’ acts as a foreign key to the ‘Movement_info table’ to represent objects that a user meets while navigating through the floor. ‘Floor_id_fk’ and ‘Building_id_fk’ are foreign keys from ‘Floor_info’ table and ‘Building_info’ table to locate where the object is placed. ‘Image_description’ column contains image-processed data about

identified objects such as their appearance and color. ‘Category’ column stores the type of object whether it is a chair, a table and etc. Both ‘Image_description’ and ‘category’ columns are filled after processing the video stream data. ‘X_coordinate’ and ‘Y_coordinate’ columns track the coordinates of the object’s current location. These two coordinates are collected via the localization information obtained via the mobile phone. If the object changes its location, X and Y coordinates will be changed.

Table 3.6: Data table of Object_info

Object_id	Floor_id_fk	Building_id_fk	image_desc ription	Category	X_coordinate	Y_coordinate

Column description:

Object_id: Primary key. A unique entry to identify each object

Floor_id_fk: Foreign key which refer to Floor_info table

Building_id_fk: Foreign key which refer to Building_info table

Image_description: String value to describe characteristics of identified object

Category: Category of object as string value

X_coordinate: X coordinate of the object location- real data type

Y_coordinate: Y coordinate of the object location- real data type

g) Movement_info table

Table 3.7 is used to collect movement information when a user navigates through a floor. ‘M_id’, ‘Object_id_fk’, ‘Path_history_id_fk’, ‘Walking_distance’, ‘Angle’, ‘Direction’, ‘Time_Stamp’, ‘X_coordinate’, ‘Y_coordinate’ are the columns available in the ‘Movement_info table’. ‘M_id’ is the unique entry to identify each user movement while ‘Object_id_fk’ is a foreign key from the ‘Object_info’ table to keep track of the objects passed by the navigator while travelling. ‘Path_history_id_fk’ is a foreign key from the ‘path_history_info’ table to identify the path used for navigation. ‘Walking_distance’ column keeps track of the distance travelled by the vision impaired individual. The angle column is

filled with the support of IMU data of the turns taken by the vision impaired individual while travelling. Angle calculation is based on the method described in figure 3.8.

- Let us imagine the user moved from position 1 to position 2 as illustrated in the picture below. The Θ value/angle can be calculated as follows.

Calculation:

$$\tan \Theta = (10 - 7) / (8 - 4)$$

$$= \tan^{-1} [0.75]$$

$$= 36.869^\circ = 37^\circ$$

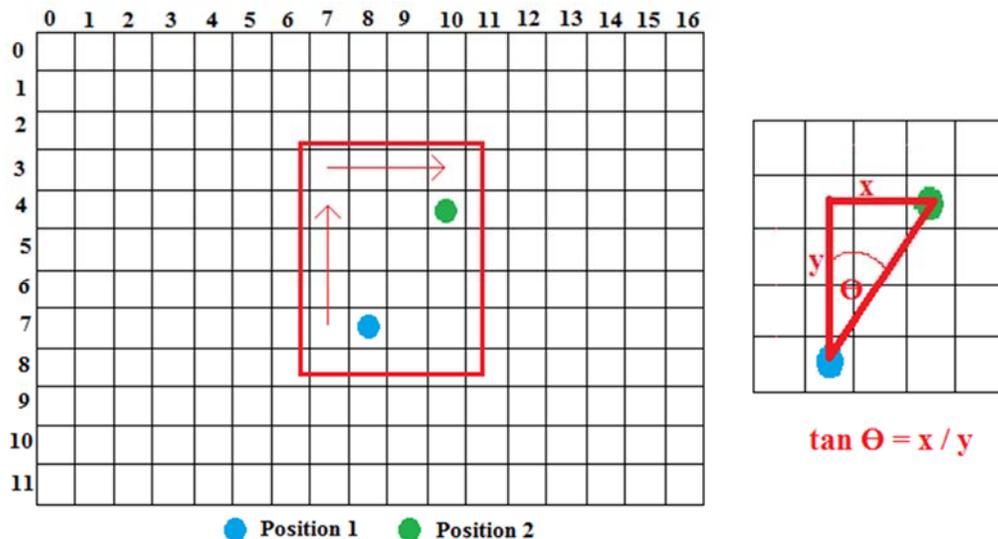


Figure 3.8: Method of calculating angle

‘Direction’ column stores the direction according to the sensor data collected via the vision impaired individual’s mobile device. ‘Timestamp’ column keeps track of the time related to every movement of the vision impaired individual, measured via the system date of the mobile device. ‘X_coordinate’ and ‘Y_coordinate’ columns track the current location of the vision impaired individual.

Table 3.7: Data table of Movement_info

M_id	Object_id_fk	Path_history_id_fk	Walking_distance	Θ	R	Time_Stamp	X_coordinate	Y_coordinate

Column description:

M_id: Primary key. A unique entry to identify each object

Object_id_fk: Foreign key which refer to Object_info table

Path_history_id_fk: Foreign key which refer to Path_history_info table

Walking_distance: Measure the distance travelled by the navigator

Angle (Θ): Calculate the angle of navigation in taking turns

Direction(R): The direction of navigation

Time_Stamp: The exact time that the cloud DB is going to receive and finish the location signal

X_coordinate: X coordinate of the object location- real data type

Y_coordinate: Y coordinate of the object location- real data type

h) Direction_info table

Direction_info table' (Table 3.8) is used to collect and store path information of a particular object relative to a given user. It consists of columns such as 'Direction_id', 'Object_id', ' Θ _value' and 'Distance'. The 'Direction_id' acts as a primary key, which uniquely identifies the object details. 'Object_id_fk' is a foreign key from 'Object_info' table that keeps track of the objects that are passed by the navigator while travelling. ' Θ value' is the angle of the object's surface point relative to the user, and 'Distance' is the distance to the object's surface point relative to the user. Both Θ value and distance (D) can be obtained after processing the video stream data.

Table 3.8: Data table of Direction_info

Direction_id	Object_id_fk	Θ Value	Distance (D)

Column description:

Direction_id: Acts as the primary key, which uniquely identifies the direction of object details receive

Object_id_fk: Foreign key, which refer to Object_info table

Θ Value: The angle of the object's surface point relative to the user

Distance (D): Distance to the object's surface point relative to the user

i) Label_info table

Table 3.9 stores 'Label_info'. It contains the label information of specific objects and places visited by the vision impaired individual. The Label_info' table consists of columns such as 'Label_id', 'Floor_id_fk', 'Building_id_fk', 'object_id_fk', 'Label_name', 'Place_description', 'StartingX', 'StartingY', 'EndingX' and 'EndingY'. 'Label_id' acts as a primary key to identify each label uniquely. 'Floor_id_fk' and 'Building_id_fk' are foreign keys from 'Floor_info' table and 'Building_info' table that figures out the location of a specific place/object. (i.e. Lab, Room, and Corridor). 'Object_id_fk' is a foreign key from 'Object_info' table to find out object details. 'Label_name' contains the name of specific place or object, which can be used for labeling purposes in map generation. 'Place_description' describes the place that this specific label belongs to. 'StartingX' and 'StartingY' are the coordinates for the starting location of the specific place or object while 'EndingX' and 'EndingY' are the coordinates for the ending location of the specific place or object. These four coordinates are collected as localized information via mobile phone sensors.

Table 3.9: Data table of Label_info

Label	Floor	Building	Object	Label	Place_	StartingX	StartingY	Ending X	EndingY
_id	_id_fk	_id_fk	_id_fk	_name	description				

Column description:

Label_id: Acts as the primary key that identifies the label details

Floor_id_fk: Foreign key that refer to Floor_info table

Building_id_fk: Foreign key that refer to Building_info table

Object_id_fk: Foreign key that refer to Object_info table

Label_name: Name of the specific place (Point of Interest)

Place_description: Describes the place that this specific label belongs to

StartingX: Starting X coordinate of the place/object location- real data type

Starting Y: Starting Y coordinate of the place/ object location- real data type

EndingX: Ending X coordinate of the place/object location- real data type

EndingY: Ending Y coordinate of the place/ object location- real data type

j) Wall_info table

Table 3.10 stores information of the walls within a building. It consists of columns such as 'Wall_id', 'Floor_id_fk', 'Building_id_fk', 'StartingX', 'StartingY', 'EndingX' and 'EndingY'. 'Wall_id' is the primary key of the table, which identifies each wall uniquely. 'Floor_id_fk' and 'Building_id_fk' are foreign keys from 'Floor_info' table and 'Building_info' table that figures out the specific location of the wall. 'StartingX' and 'StartingY' are the coordinates that mark the starting location of the wall whereas 'EndingX' and 'EndingY' track the coordinates of wall's ending location. These four coordinates are collected as localized information via the mobile phone with the help of an accelerometer and gyroscope sensors.

Table 3.10: Data table of Wall_info

Wall_id	Floor_id_fk	Building_id_fk	StartingX	StartingY	EndingX	EndingY

Column description:

Wall_id: Acts as the primary key, which identifies each wall uniquely

Floor_id_fk: Foreign key, which refer to Floor_info table

Building_id_fk: Foreign key, which refer to Building_info table

StartingX: Starting X coordinate of wall - real data type

StartingY: Starting Y coordinate of wall- real data type

EndingX: Ending X coordinate of wall- real data type

EndingY: Ending Y coordinate of wall -real data type

k) Door_info table

‘Door_info table’ (Table 3.11) is used to store information on doors within the building. The Door_info table consists of columns such as the ‘Door_id’, ‘Floor_id_fk’, ‘Building_id_fk’, ‘Wall_id_fk’, ‘StartingX,’ ‘StartingY’, ‘EndingX’, ‘EndingY’, ‘Is_open’. ‘Door_id’ is the primary key of the table, which identifies each door uniquely. ‘Floor_id_fk’, ‘Building_id_fk’ and ‘wall_id_fk’ are foreign keys from ‘Floor_info table’, ‘Building_info table’ and ‘wall_info_table’ that figure out the specific location of the door. ‘StartingX’ and ‘StartingY’ are the coordinates of the door’s starting location while ‘EndingX’ and ‘EndingY’ are the coordinates for the door’s ending location. These four coordinates are collected as localized information through the mobile phone with the help of its sensors. The side from which the door is open can be obtained through image processing. However, it would take some time before it is added to the database. ‘Is_open’ column stores the status of the doors whether they are open or closed in Boolean value.

Table 3.11: Data table of Door_info

Door_Id	Floor_id_fk	Building_ id_fk	Door_ id	StartingX	StartingY	EndingX	EndingY	Is_open

Column description:

Door_id: Acts as the primary key which identifies each door uniquely

Floor_id_fk: Foreign key which refer to Floor_info table

Building_id_fk: Foreign key which refer to Building_info table

Wall_id_fk: Foreign key which refer to Wall_info table

StartingX: Starting X coordinate of door - real data type

StartingY: Starting Y coordinate of door- real data type

EndingX: Ending X coordinate of door - real data type

EndingY: Ending Y coordinate of door -real data type

Is_open: Boolean value to store door status – Door open or close

All the above-mentioned tables are requisites to generate a detailed real-time map. Data for the tables can be stored with the help of optimization mechanisms described in section 3.3.2.2.

3.3.2.2 Database Optimization Techniques Used in Performance Enhancement

Research on optimizing database systems have been an ongoing process within the past few years [55]. With the growth and enhancement of applications, data queries are becoming increasingly complex, thus creating a need for efficient data retrieval mechanisms.

The performance of a database is determined by both internal and external factors. Database settings and structure, indexing and implementation are the most common factors that determine the performance of the database internally. If the internal factors are not configured

appropriately, it would degrade the performance of the database in retrieving data. Such situations can be mitigated by the use of five common techniques mentioned below [64].

a) *Use an external cache.*

PostgreSQL memories are independent as they do not offer single synchronize-level memory management [65]. Therefore, preloading or caching the table with PostgreSQL is incompatible. Caching is possible with external tools such as BIMcache. Pgmemcache is a set of PostgreSQL user-defined functions (API's) that provide an interface to BIMcache. As a pre-requisite, Pgmemcache recommends the implementation of libmemcache. Installation of BIMcache could be used to enhance the performance of the database. Hence, the simulation environment implemented in this research adopts BIMcache to optimize performance.

Figure 3.9 illustrates the overview of BIMcache. A user can generate a real-time map based on the X,Y coordinates of previous user movements without constantly accessing the database. Loading the real-time map using BIMcache, increases the efficiency of the AccessBIM model as it reduces the time taken to generate the map.

Figure 3.10 illustrates the output of BIMcache. Most recent navigations could be loaded into the movement_info table which facilitates real time map generation without collecting data from the simulator

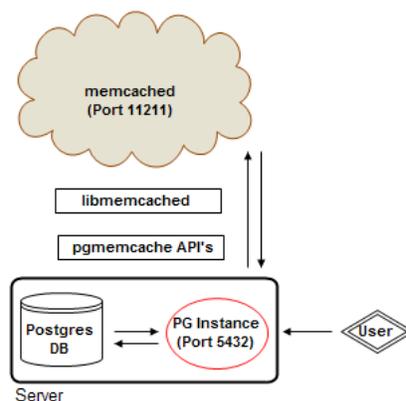


Figure 3.9: Overview of BIMcache (Taken from: <http://raghavt.blogspot.com/2011/07/pgmemcache-setup-and-usage.html> [65])

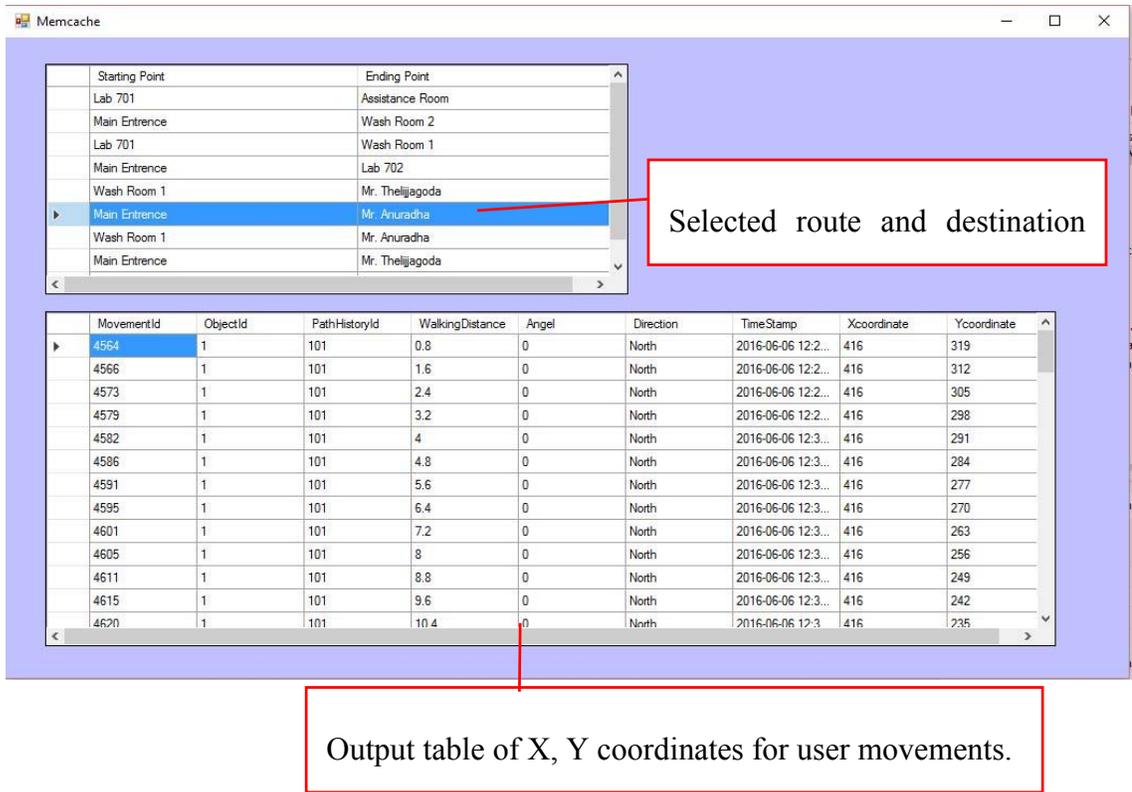


Figure 3.10: Output results of BIMcache

b) Indexing strategy.

A database index allows a query to efficiently retrieve data from a database. Indexes are related to specific tables and can have more than one index built from it [66]. However, it is only beneficial if the number of rows to be retrieved from a table is comparatively small [67].

- Partial index

A partial index is a subset of a table's data which uses the WHERE clause to increase the efficiency of the index by reducing its size whilst, making it easier to maintain and faster to scan as it takes less storage [67].

- Expression index

Expression indexes are indexes on a function or scalar expression computed from one or more columns of the table. PostgreSQL allows indexing the results function and it leads to obtaining

fast access to tables based on results of the location computations through rapid random lookups. Indexes on expression can be used to enforce constraints that are not definable, unique and simple [68]. This index is useful when the retrieval speed is more important than the speed of insertion and updating. For instance, finding a given date by date cast value.

- B -Tree index

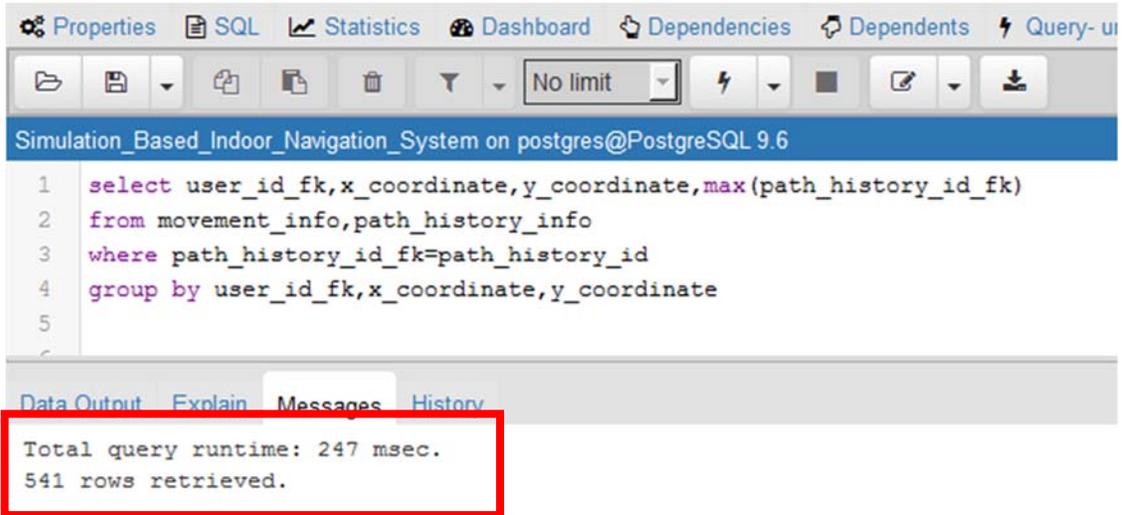
B-Tree index entries are sorted in ascending order by default, which supplies a different sort order for an index in some cases. Since designed table data is querying the table in sorted order by limiting the result, it is possible to reap benefits by creating an index in the same order. B-tree indexes are used with single column indexes when required “nulls to sort last” behavior [67]. The AccessBIM model utilizes fine tuning and database optimization features of B-Tree index.

c) Query rewriting

Base tables of AccessBIM model contains a large amount of data regarding building characteristics and user movements. Real-time map generation only needs data related to the most recent movements in given routes. Hence processing a large number of non-relevant data for map generation is expensive and time-consuming. This research work rewrites the basic SQL queries to filter out the most relevant details for map generation from a large data set.

SQL JOIN clause is able to optimize the queries by combining rows from two or more tables, based on the common field between them.

i.e.: Obtaining a real-time map would require data from both movement_info and path_history_info tables. Figure 3.11 demonstrates a situation where a real-time map is generated without query rewriting.



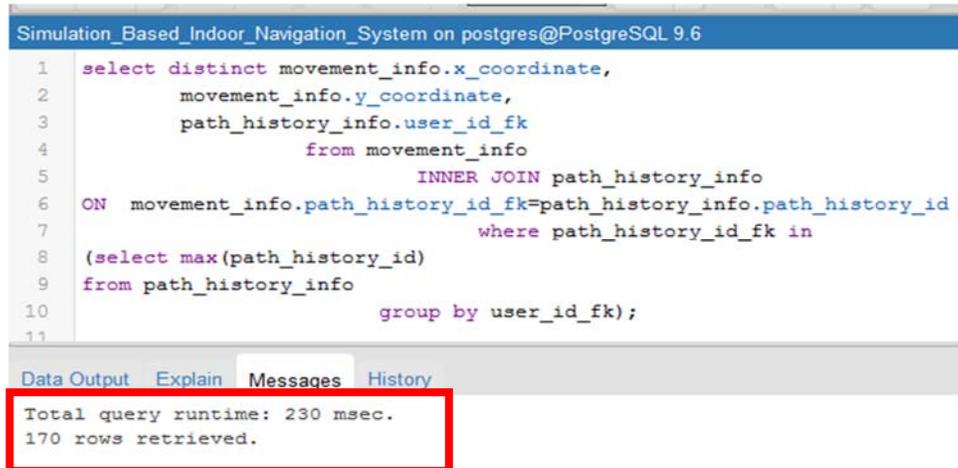
```
Simulation_Based_Indoor_Navigation_System on postgres@PostgreSQL 9.6

1  select user_id_fk,x_coordinate,y_coordinate,max(path_history_id_fk)
2  from movement_info,path_history_info
3  where path_history_id_fk=path_history_id
4  group by user_id_fk,x_coordinate,y_coordinate
5
6

Data Output Explain Messages History
Total query runtime: 247 msec.
541 rows retrieved.
```

Figure 3.11: Output of query execution without SQL JOIN

Figure 3.12 is an example for a situation which uses query rewriting by the use of a SQL JOIN. Highlighted sections of the figure prove that the SQL JOIN condition is able to filter and retrieve the most relevant details for map generation within a shorter time.



```
Simulation_Based_Indoor_Navigation_System on postgres@PostgreSQL 9.6

1  select distinct movement_info.x_coordinate,
2                 movement_info.y_coordinate,
3                 path_history_info.user_id_fk
4                 from movement_info
5                 INNER JOIN path_history_info
6 ON  movement_info.path_history_id_fk=path_history_info.path_history_id
7                 where path_history_id_fk in
8 (select max(path_history_id)
9  from path_history_info
10                 group by user_id_fk);
11

Data Output Explain Messages History
Total query runtime: 230 msec.
170 rows retrieved.
```

Figure 3.12: Output of query execution with SQL JOIN

d) Use of stored procedures

A stored procedure is a prepared SQL code that can be reused many times without repeatedly writing the query. This research project adopts stored procedures to implement centralized logic in the database instead of implementing it on the application aspect to gain following benefits [52].

- Compiles stored procedure once and then reutilizes the execution plan to provide efficient performance boots.
- Reduces traffic between AccessBIM model and the test simulator by reducing SQL queries into single line to transmit over the wire. This would become a major concern when the project is mapped in real as the actual environment would be used to collect data using mobile devices.
- Allows to have multiple input and output parameters to generate real-time map while UDF allows only one input parameter at once.
- Permits to use DML statements like insert, update and delete while UDF permits to use only the selected statements.
- Transaction can be done through the stored procedure where UDF is unsupported.
- Minimizes round trips between the database and application to save execution time by enfolding all SQL statements inside a function stored in PostgreSQL database server.

e) Changes to the schema

The interview done with “The Employers' Federation of Ceylon” identified the tables which should be included in the AccessBIM framework. Based on the discussion with them, the following steps were examined with the main normal forms to ensure data consistency.

1. All the tables are designed with a unique primary key and all the columns are designed to contain only one value to make sure that the database is in the first normal form.

2. Data tables are designed with a single column primary key according to the requirement of data inside the table, therefore the table designed is also in the second normal form.
3. Converting all the tables in to the third normal form by removing all the transitive dependencies where attributes depend on the non-primary key.
4. The database design does not contain any super key or candidate key, hence no BCNF checking is needed.

3.3.3 Test Environment Implementation Approach

According to a research on map generation [19, 20], complex and interacting parameters influence the real-time map generation process whilst, a number of sensor data are required to be collected at the same time in order to develop an accurate output. The research work presented in this thesis has mapped an actual indoor environment into a virtual environment for testing purposes. The reasons for using a virtual environment (computer simulation) is to reduce, the building cost of a controlled environment, time associated in creating a new system and to reduce the complexity in data collection. Computer simulation together with statistical analysis techniques has the ability to support decision-making by generating a similar environment to the actual scenario. Speedy analysis in data retrieval and the detection of problems, bugs and difficulties in map generation are the major benefits of implementing a computer simulation before the actual implementation.

3.3.3.1 Test Simulator Design

R. McHaney stated that computer simulation is used as a model to draw conclusions, providing awareness on the characteristics of real world elements being studied, with the use of computer programming [69].

The author used a test simulation engine as it has the ability to collect and submit environmental data while a vision impaired individual moves within the simulated floor. This simulator includes capabilities such as the ability to:

- Provide localization information – Localization data represent X,Y coordinates of the reference point of the location. It provides meaningful information to create the map.
- Provide environmental characteristics - The simulation is based on output data obtained from the spatial environment that contains distance (R) and angle (Θ) relative to the users' current location when they walk within a building and image processing data that provides details on obstacles and surrounding details [20].
- Collect and transfer data – Simulator is designed to collect spatial information and image processing data of the desired location while the user walks within the simulated environment.

This test simulator module and the simulation environment are used to indicate that AccessBIM framework is capable of storing and generating a real time map with crowdsourced data.

3.3.3.2 Requirements of Test Simulator Design

The literature survey on indoor navigation, database optimization and real-time map generation [14, 15, 41, 70] identifies the functional requirements implemented in a test simulator.

Functional requirements implemented in the test simulator:

- Capability of the AccessBIM framework in real-time map generation.
- Facilitates navigation without others interaction.
- Collects crowdsourced data in a controlled environment to reduce noise in data collection.
- Builds a fixed communication channel with the data collector and the database server where AccessBIM framework is located.
- Reduces time spent on data collection.

One of the major reasons behind utilizing a simulator to collect data is the fact that the work presented in this thesis is dependent on the inputs of the other researchers of the same research

group. Hence, the use of simulator allowed the author to complete his part of the project without relying on the input of other researchers.

Given below are the quality attributes implemented in the test simulator:

- Availability- The test simulator should be up and running to any user at any given time.
- Performance- Multiple users would be able to use it simultaneously at any given time.
- Reliability- Ability to provide the expected data accurately and precisely under any condition.
- Security - Prevent unauthorized modifications to already saved maps.

Once the primary requirements are recognized and categorized into functional and quality attribute aspects, a detailed description is obtained for each of the functionalities. These requirements acted as the base of the research project.

3.3.3.3 Implementation of Test Simulator Environment

The design phase describes how the system works with hardware, software, networks, databases etc. Therefore, this research created a simulation that is an accurate representation of the building. The simulator acts as a virtual environment and provides facilities to collect indoor environment details while users navigate within the simulator.

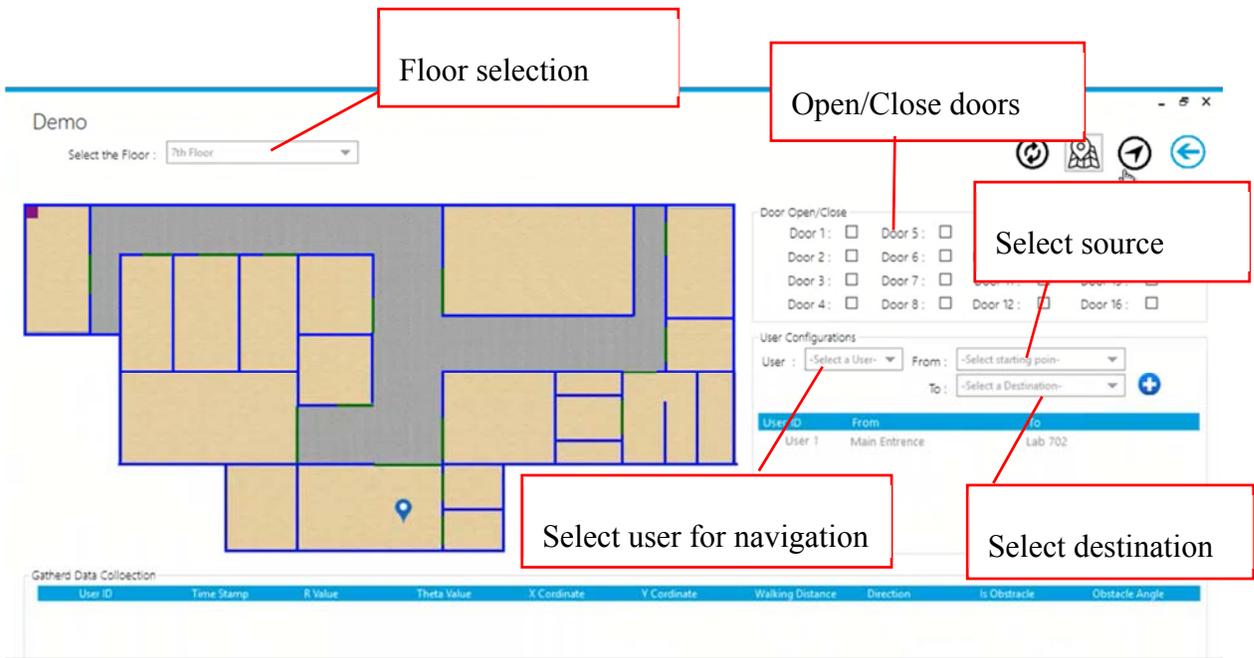


Figure 3.13: Created simulator (Virtual environment)

Figure 3.13 is a diagram of the test simulator. As an example of a navigation, after configuration, users can navigate to the desired destination and the data collected and submitted to the ‘Movement_info’, ‘Object_info’, ‘Door_info’ and ‘Wall_info’ tables are used to generate a real-time map. Figure 3.14, 3.15, and 3.16 illustrate the behavior of simulator when a user navigates.

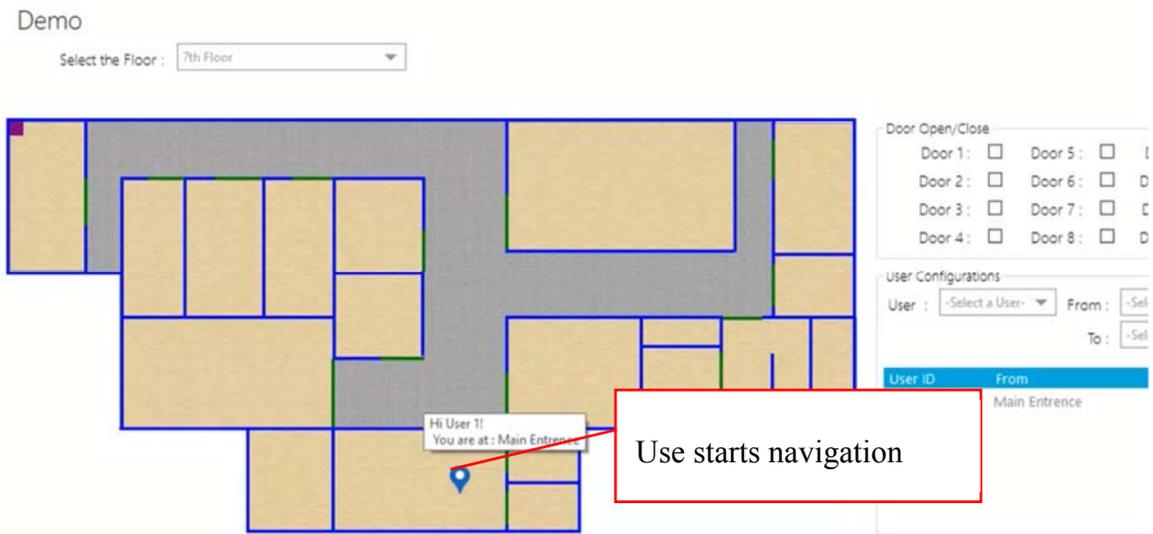


Figure 3.14: User starts navigation.

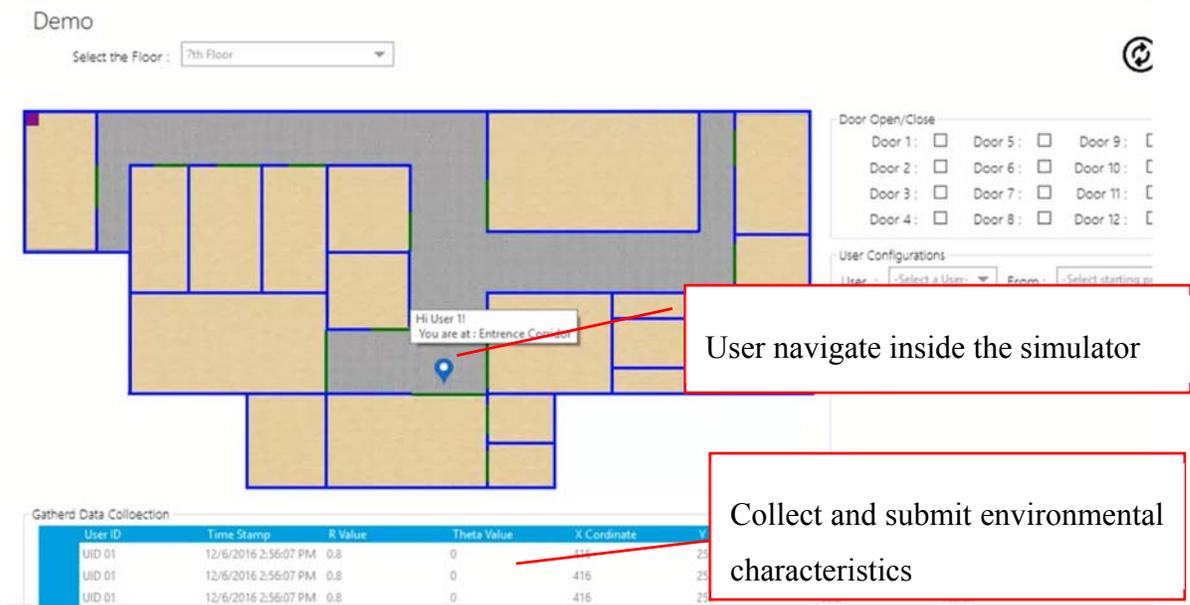


Figure 3.15: User navigation and data collection

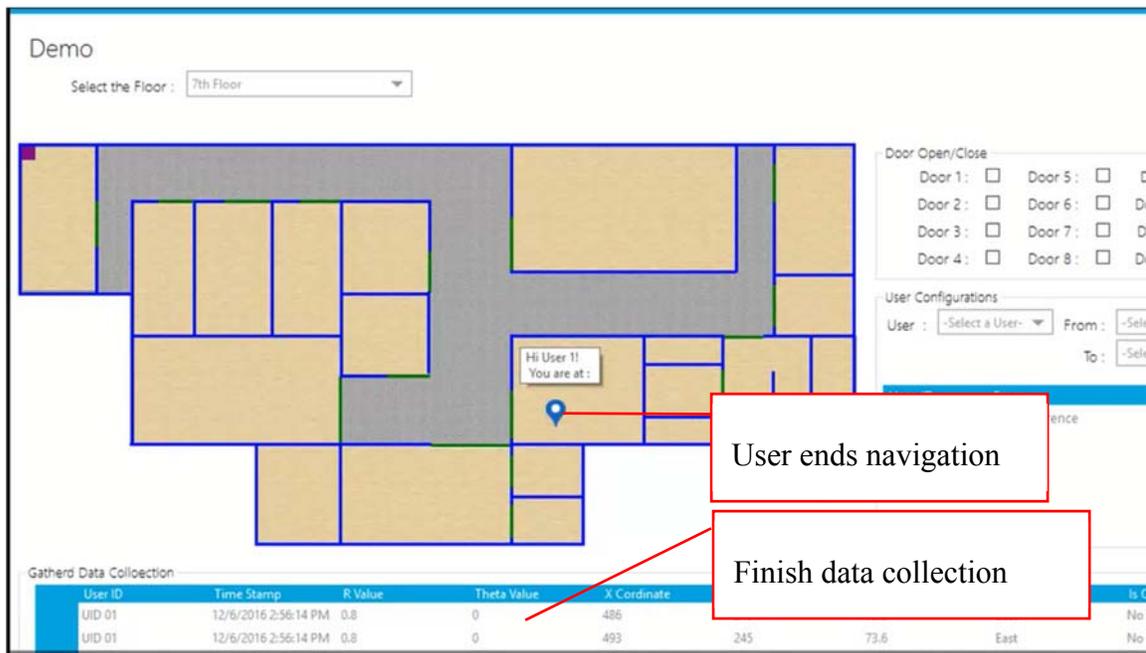


Figure 3.16: User finish navigation and data collection

Data gathering, storing and retrieving happens through c# API, which is illustrated using figure 3.17.

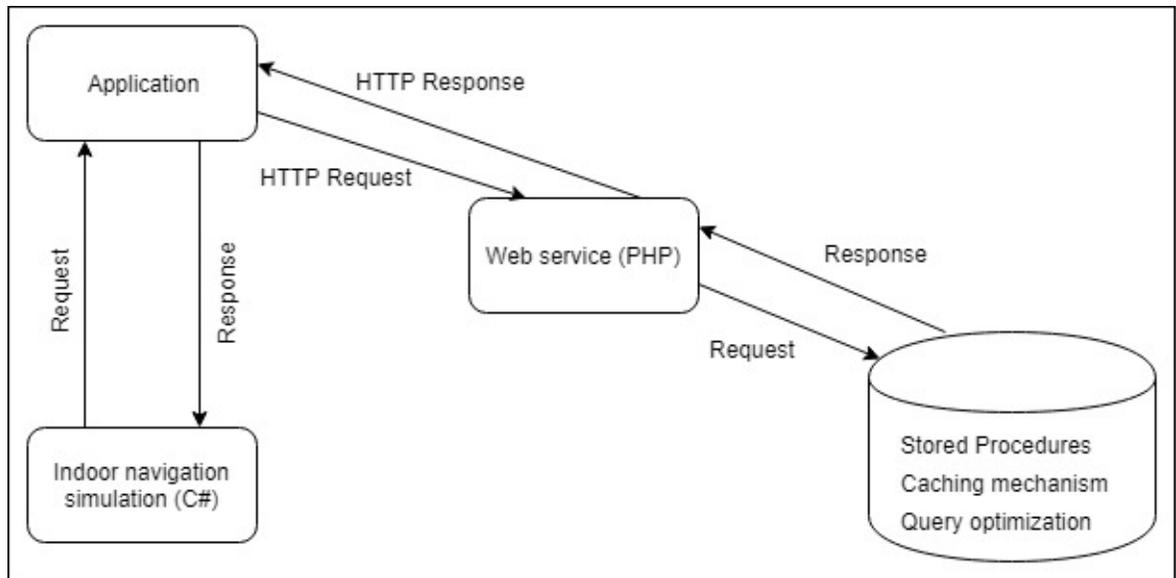


Figure 3.17: Overview of simulated test Engine

The test engine consists of three major components. Simulation, database connectivity and the database. Stored procedures, caching mechanisms, change of the database design, indexing, query rewriting and functions and triggers are used to collect environmental changes through the test engine to store data within the database in an optimized manner.

As shown in figure 3.17, C# application (simulator) and PostgreSQL database are connected with the web service that is based on REST (**RE**presentational **S**tate **T**ransfer) architecture, which is used to transfer data between the C# application and the PostgreSQL database. REST is usually faster and leads to lower bandwidth.

API was derived using C# dll, while the web service was written using PHP and PostgreSQL database management system. The created dll consists of the database connector and the API functions (insert, update, and delete data within the table) to send and receive required information. The data should be received by the database in real-time without any delay. Hence, a JavaScript Object Notation (JSON) object is used to speed up the process. Main queries are executed inside the PostgreSQL database management system by using stored procedures so that a number of programs can share it.

a) *Collect data from test simulator*

The simulator allows selecting a number of users, their starting point and destinations respectively. Once the users' starts navigating, the movement information of every user will be collected. Following variables will be captured during the navigation process.

- i. Location coordination

The floor plan of the simulator holds the following scale with the real world.

Map ratio : **8 pixel = 0.8 m**

Therefore, every time a marker passes eight pixels, it will be counted as 1 step. Step count will be captured according to that scenario.

- ii. User's angle. (Θ value)

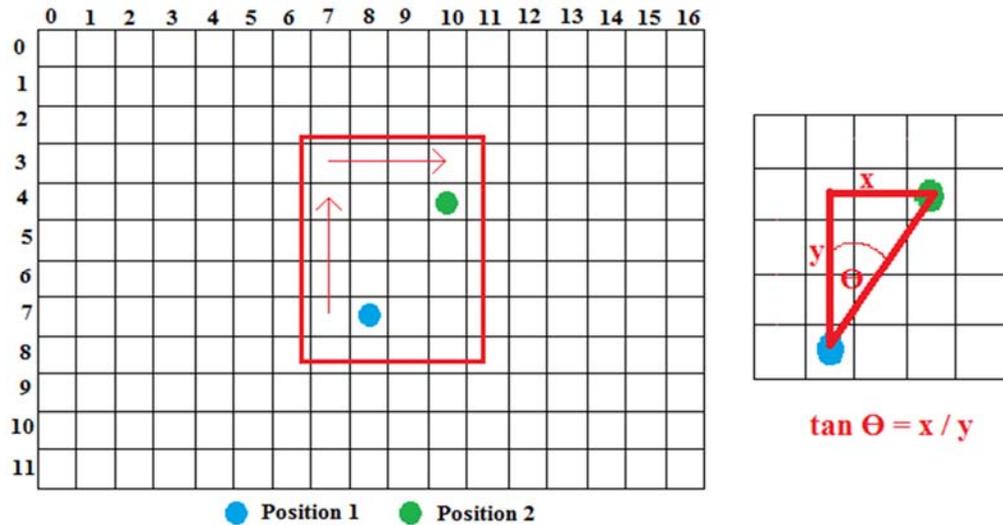


Figure 3.18: Identify user's angle

Let's say the user moved position 1 to position 2 as mentioned in figure 3.18. Then the Θ value will be calculated as follows.

Calculation:

$$\begin{aligned}\text{Tan } \Theta &= (10 - 7) / (8 - 4) \\ &= \tan^{-1} [0.75] \\ &= 36.869^{\circ} \\ &= 37^{\circ}\end{aligned}$$

iii. Obstacle detection¹

Once a user navigates within an area after selecting the required source and destination, the simulator is able to collect details regarding the surroundings including walls, doors and objects. If there is an object within the vision impaired individual's area of interest, it will be identified as an obstacle and the distance and the angle will be calculated based on the user's current position. At the same time, that information will be sent to the database as well. If the vision impaired individual is able to go across objects occasionally it could be named as a movable obstacle (i.e. - door). There can be multiple movable obstacles within the vision impaired individual's area of interest. Yet, it identifies each individual object and updates its status in the database (i.e. whether the door is open or not)

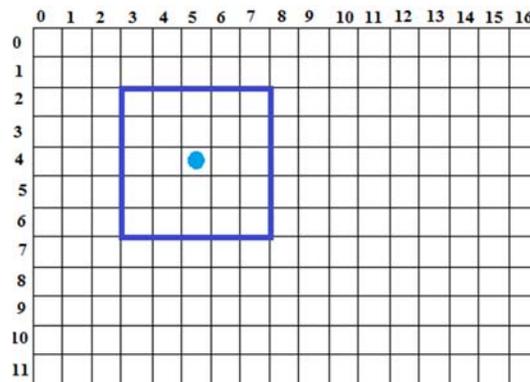


Figure 3.19: Obstacle detection

¹ Obstacle detection is a component of other member of same research group based on image processing [20].

Although obstacle detection is beyond the scope of this thesis, obstacles could be detected by identifying the x,y coordinates of the navigator and then searching for obstacles that are within a radius of 5m relative to the user's position as illustrated in figure 3.20.

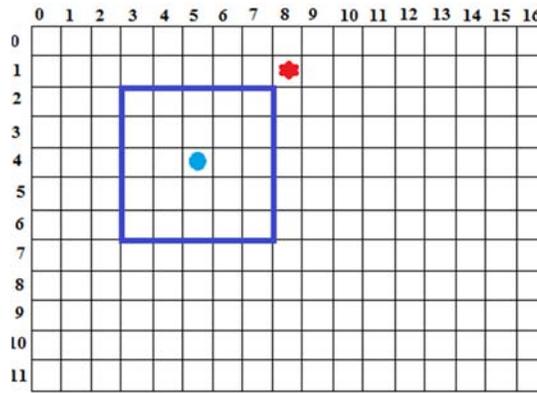


Figure 3.20: Identify non-movable obstacle

b) Data communication between simulator and the server

The communication between the server and the navigation simulator is done using JSON; because of its ability to multicast the same message to multiple devices simultaneously.

3.4 Real-Time Map Creation

An accurate indoor map plays a vital role in a navigation system; hence the data that is represented in the indoor map must be accurate and up-to-date. In order to generate an indoor map that facilitates vision impaired navigation, an experimental map was derived using C#. Successful navigation inside the simulator engine enabled to collect simulated crowdsourced data obtained via smart phones, IoT devices, image processing and GAIT analysis into a database to generate a real-time map to aid navigation. The overview of the map generation process is presented in figure 3.21.

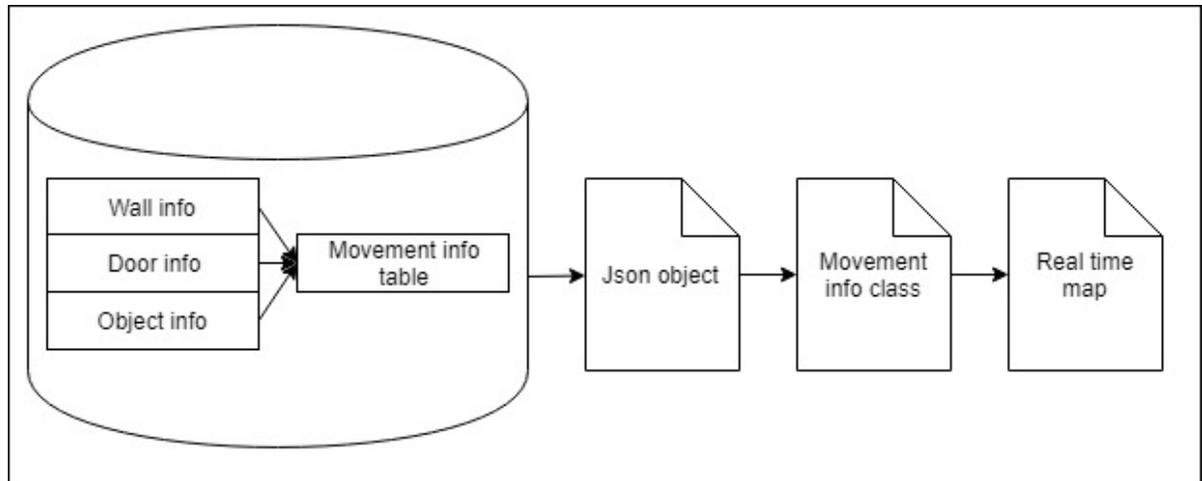


Figure 3.21: Overview of map generation

The above diagram (Figure.3.21) illustrates how the data is retrieved from the database to build the real-time map. Door information, wall information, object information and label information could be gathered while the user navigates within the indoor environment in real world scenario. However, in the simulation, the database contains “wall_info”, “door_info”, “object_info” tables with manually entered data. As a result, the user who navigates using the simulator is able to recognize his current location and environmental characteristics related to his location. Moreover, it can send them to ‘Movement information’ table as well as update the door status and location of the objects in ‘Object information table’. The Simulator is able to extract the table data through JSON object and build three classes called ‘wall class’, ‘door class’, and ‘object class’. Created classes can be used to generate an automatic real-time map based on the collected crowdsourced data. Real-time map generation is based on the following data extracted from the above-mentioned classes. Wall information is related to the current x,y coordinates of the user.

- Door information related to current x, y coordinate of the user
- Status of movable obstacles (whether the door is open or closed. If the door is closed it is represented in red and if the door is closed it would be represented in green)
- Locations of the objects according to the user’s current x, y coordinates.

The real-time map generation process needs to communicate with the AccessBIM database from time to time to track environmental characteristics. The data server and the simulation

application are built based on client-server architecture. The map and the navigation pointer are displayed on the client's smart device. All data in the actual environment will be collected through the sensors in smart devices, which then generates the routing data from the virtual environment in to a desktop application. The desktop application and mobile devices exchange data through a communication protocol. The sensor information is partially processed by the application and sent to the database server to build the map. Figure 3.20 illustrates a partially generated map after collecting data from the test simulator. (Routes of the most recent navigations are drawn in dashed lines in figure 3.22.)

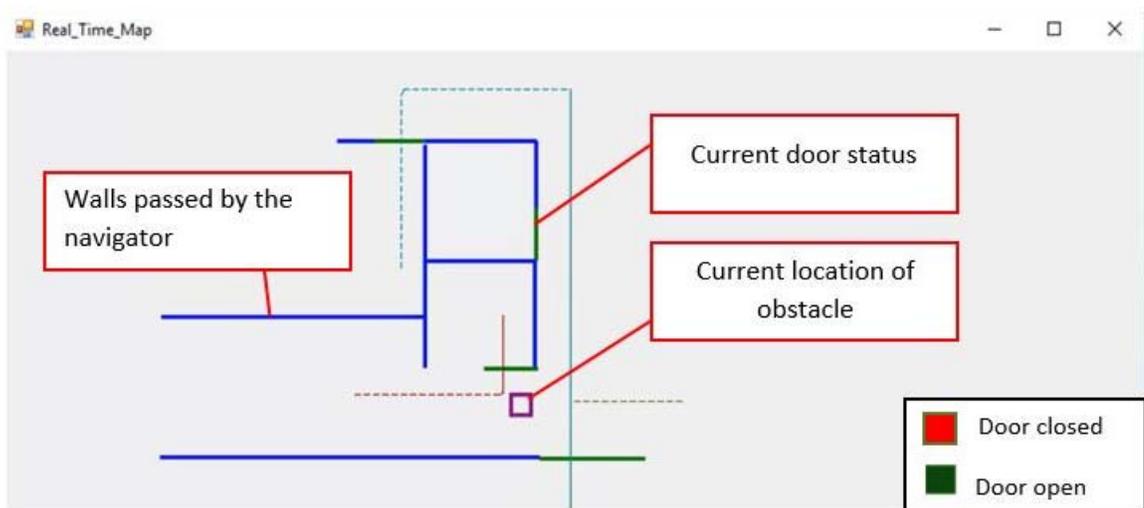


Figure 3.22: Generated real-time map according to multiple user navigations

3.5 Chapter Conclusion

This chapter addressed the experimental objectives of the research and the design and implementation of the AccessBIM framework in detail. The AccessBIM framework consists of five layers. The data collection layer collects crowdsourced data which is then sent to the Database through the API in the User Interface layer. In the functional layer, the database connection is initiated, called for function and returned, before the connection is terminated. Efficient information processing occurs in the Database optimization layer due to query optimization and indexing, while the backend layer filters only the required data to generate the map.

The AccessBIM database consists of 11 relations and several database optimization techniques such as indexing, query rewriting and stored procedures have been applied to enhance performance. A test simulation engine was used as it reduces the cost of building a controlled environment apart from reducing the time associated in creating a new system. The test scenarios described in section 5.2 demonstrates the accuracy of the simulator as an accurate real-time map is generated a within few milliseconds.

Chapter 4 – Map Construction Algorithms

4.1 Chapter Overview

This chapter discusses the four fundamental algorithms involved in the research. Section 4.2 provides an overview of the four algorithms while section 4.3 describes the algorithm for indoor real-time map generation. Section 4.4 describes the database optimization algorithm and section 4.5 discusses the algorithm for BIMcache optimization. The algorithm for crowdsourced data collection is described in section 4.6 while section 4.7 concludes the chapter.

4.2 The Four Algorithms

The algorithmic component stands out as the core of the research as it differentiates the AccessBIM framework from other indoor navigation systems. The AccessBIM database is equipped with four algorithms namely: ‘The Database Optimizer Algorithm’, ‘The BIMcache Optimizer Algorithm’, ‘The Crowdsourced Data Collection Algorithm’ and ‘The Real-Time Map Generation Algorithm’. Most research on indoor navigation do not focus on database optimization [14, 15, 37] while the ones that do, do not publish the algorithms openly [55, 57]. Hence, the researcher developed several algorithms that would optimize the performance of not only indoor navigation related databases, but any database that stores spatial information.

The Crowdsourced Data Collection Algorithm identifies the location of each user and obstacle on a periodic basis, in the form of x and y coordinates while the Database Optimizer algorithm applies stored procedures and query rewriting to the relations in the database. The BIMcache Optimizer Algorithm acts similar to a caching mechanism by searching the database for already existing paths thus saving time and the cost of searching data. The synthesis of these three algorithms facilitate the Real-Time Map Generation Algorithm to generate a meaningful indoor map with minimum consumption of resources.

4.3 Algorithm for Indoor Real-time Map Generation

The map generation is an integral part of the algorithmic component in the entire process. The user set $U = \{U_1, U_2, U_3, \dots, U_i, \dots, U_{n-1}, U_n\}$ who are using the system will be requesting navigation assistance to reach their desired destination 'D'. Each time a request is made, the system should provide an optimal path 'P' for the user to reach his preferred destination. The real-time map generation algorithm given in figure 4.1 is the main algorithm which a number of sub algorithms are embedded to aid to generate the optimal path.

At the beginning of the path generation process data sets and buffers are restored to the last map generated through the device. The initializing process starts soon as a user 'U' requests for navigation assistance. In order to process the user information, the x, y coordinates and special environmental information 'S' along with the destination 'D' (already saved destinations in the system) are retrieved and stored for further referencing. Before compiling the optimal path from scratch, the BIMcache is searched to find any existing, recently created paths for the same destination via the user coordinates x, y of user U_i . If a match is found, that path information will be loaded to the user application for processing.

If a match is not found in the BIMcache, the path generation request is sent to the database for further processing. Here, the path would be generated based on the real-time information obtained from the environment. The crowdsourced data gathering algorithm which is further explained in section 4.6, gathers user data periodically, in order to generate the optimal path. At this point, the path is generated in a way that it will include any obstacles within the user's block space by filtering the objects within the defined block size 'r' of the user ' U_i ' and periodically updating any user generated information into the database. An optimal map showing the shortest and the most convenient path is filtered out from the database's map information which constitutes the user U_i 's block size 'r'. Based on the filtered information, the optimal path is loaded to the user application for use.

When there is a change in path information due to the placement of new obstacles, the existing path information available in the BIMcache will be replaced by the new information using the `AccessBIM_Database_Optimizer_Algorithm` that is further discussed in section 4.5.

Algorithm - DYNAMIC_INDOOR_REALTIME_MAP_GENERATION (U, r, S, D)
mapping information

BEGIN Input - Crowdsourced data consisting of users $U = (U_1, U_2, \dots, U_n)$ where x is user U_n 's X-coordinate, and y is user U_n 's Y-coordinate which provides user location. ObjX is the X-Coordinates to the object for user U_n , and ObjY is the Y-Coordinates to the object for user U_n . Block size for user is r and the time interval of data retrieval is t . S stands for the special environmental information obtained.

Output – Recursively obtain the

```

1) Initialize
2) while |U|.newRequest do {
3)   GET_REQUEST (Si, D);
4)   if (D!) then
5)     redirect_to (User_App);
6)   else {
7)     find P such that
8)       GET_BIMCACHE { opt_path (P) ∈ S,D };
9)     if (P=1) then
10)      LOAD { opt_path (P) };
11)    else {
12)      DB_QUERY { find P such that opt_path(P) ∈ S,D };
13)      for (i=1) to END do {
14)        CROWDSOURCE_DATA_GATHERING (U,x,y,Obj[O]);
15)        GET_PARAMETER ( |U| new,r );
16)        for (opt_map(S) = (db_map (U) ∩ r )) do {
17)          (opt_path (P) ∈ opt_map (S)(|U| new,Obj[Oi],r));
18)          LOAD { opt_path (S) };
19)        }
20)      }
21)    }
22) }
23) if (Obj(S)t != Obj(S)t-1) then
24)   AccessBIM_BIMCACHE_OPTIMIZER_ALGORITHM ();
25) }
```

END

Figure 4.1: Dynamic Indoor Real-Time Map Generation

Indoor map generation is a process that consists of three major parts, namely crowdsourced related data collection, database optimization and efficient data retrieval. Data was collected via a simulation environment. Collected data was stored in an optimized data model with the help of stored procedures in order to build the AccessBIM data model. Stored data is retrieved with minimal effort as proven under section 6.3.1, from the AccessBIM using stored procedures and query rewriting. Retrieved data was classified to generate a meaningful map. Figure 4.2 illustrates the workflow of the AccessBIM database optimization model in terms of a flow chart.

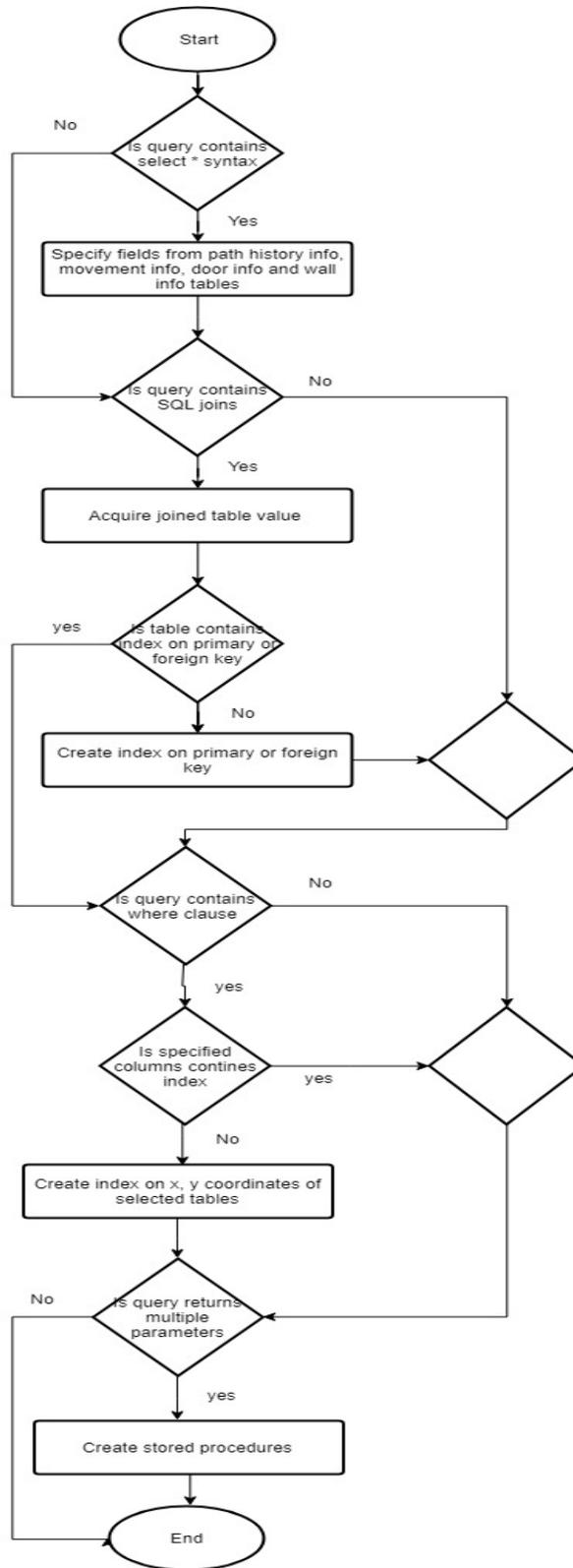


Figure 4.2: Flow of AccessBIM database optimization model

4.4 Algorithm for Database Optimization

Algorithm – AccessBIM _DATABASE_OPTIMIZER_ALGORITHM

Input – AccessBIM model consist of 11 relations $Q = \{Q_1, Q_2, \dots, Q_{11}\}$

Output – Optimized AccessBIM database model

BEGIN

```
1) if (SELECT * QUERIES) then {
2)     field(Path_history_id, Movement_id, X_coordinate, Y
   coordinate)
3)     goto(step_4); }
3) else {
4)     if (JOIN_QUERIES) then {
5)         GET_JOIN_TABLE(value);
6)         if (TABLE  $\in$  primary_key, foreign_key) then {
7)             if (WERE clause) then {
8)                 if (COLUMN  $\in$  index ) then {
9)                     if ( RETURN MULTIPLE PARAMETERS) then {
10)                        CREATE_STORED_PROCEDURE();
11)                        return; }
12)                    else
13)                        return; }
14)                    else {
15)                        CREATE_COLUMN_INDEX;
16)                        goto(step_9); }
16)                    else
17)                        goto(step_9); }
18)                else {
19)                    CREATE_PRIMARY_KEY_FOREIGN_KEY;
20)                    goto(step_7); }
21)            else {
22)                goto(step_7); }
23)        }
```

END

Figure 4.3: AccessBIM Database Optimizer Algorithm

The introduced database schema model is explained in section 3.3.2. Figure 4.1 illustrates the flow that is followed to build the optimized database model. The AccessBIM database optimizer algorithm in figure 4.3 focusses on applying stored procedures and query rewriting to the existing relations (R) in the database. The eleven relations listed in figure 3.5 of section 3.3.2.1 act as inputs to this algorithm. Each relation traverses through the algorithm to apply query rewriting and stored procedure to the exact location. The algorithm contains a set of conditions that check qualities of the relations. It has the capability of minimizing problems that occur in data creation, insertion, updating and deletion. A set of identified conditions supported the process of generating optimal data model.

4.5 Algorithm for BIMcache Optimization

BIMcache is a technique used to find the existing movement information for a particular source and destination. The use of BIMcache reduces the time taken to search data for map creation whilst updating environmental changes that occur after navigation.

BIMcache is an approach that could be used for faster data retrieval from the AccessBIM framework. The most recent navigation is added to the BIMcache for possible distinct navigation. The map generated based on the most recent navigation is loaded automatically to the user when trying to navigate to a previous destination, hence saving time and the cost of searching data. The following algorithm introduced by the researcher in figure 4.4 facilitates successful implementation and application of BIMcache.

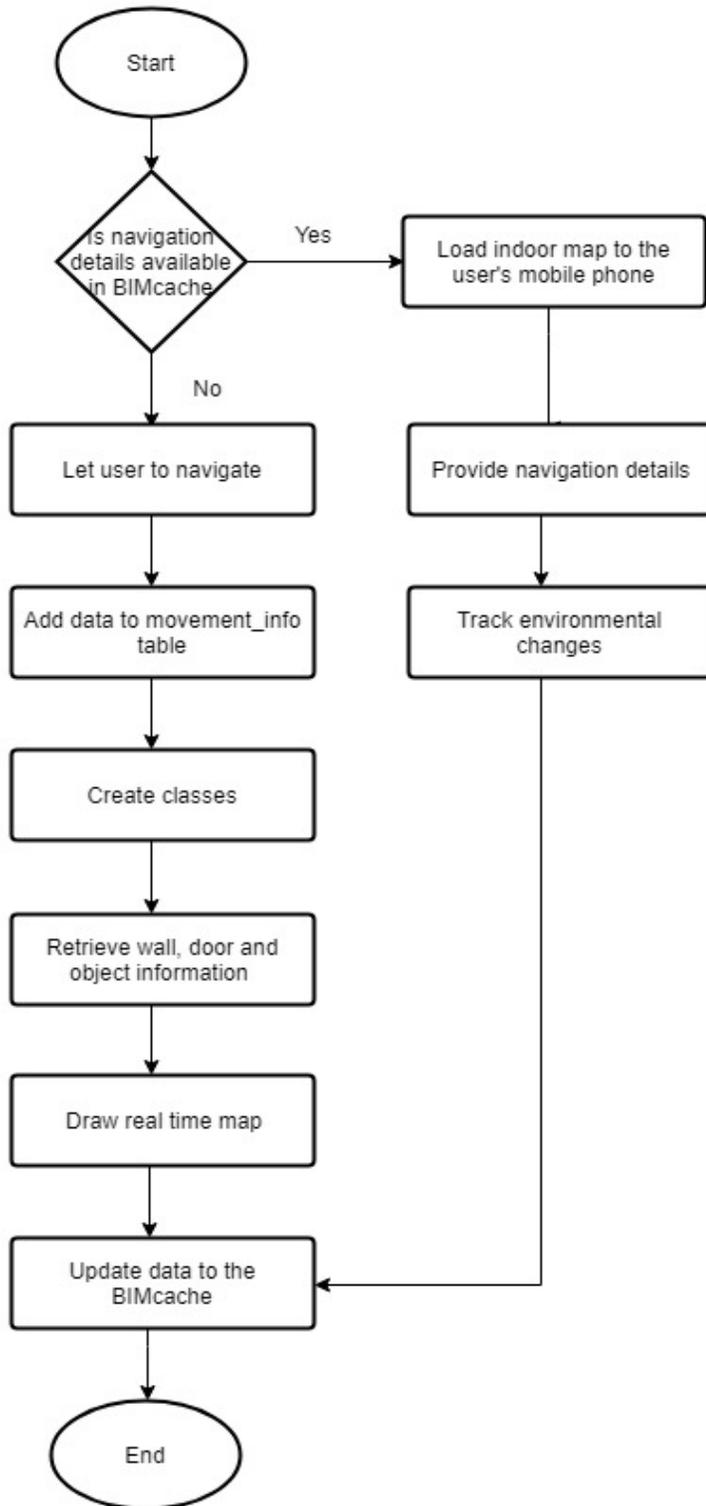


Figure 4.4: Flow of BIMcache time reduction model

Figure 4.5 illustrates the BIMcache optimizer algorithm for rapid data access.

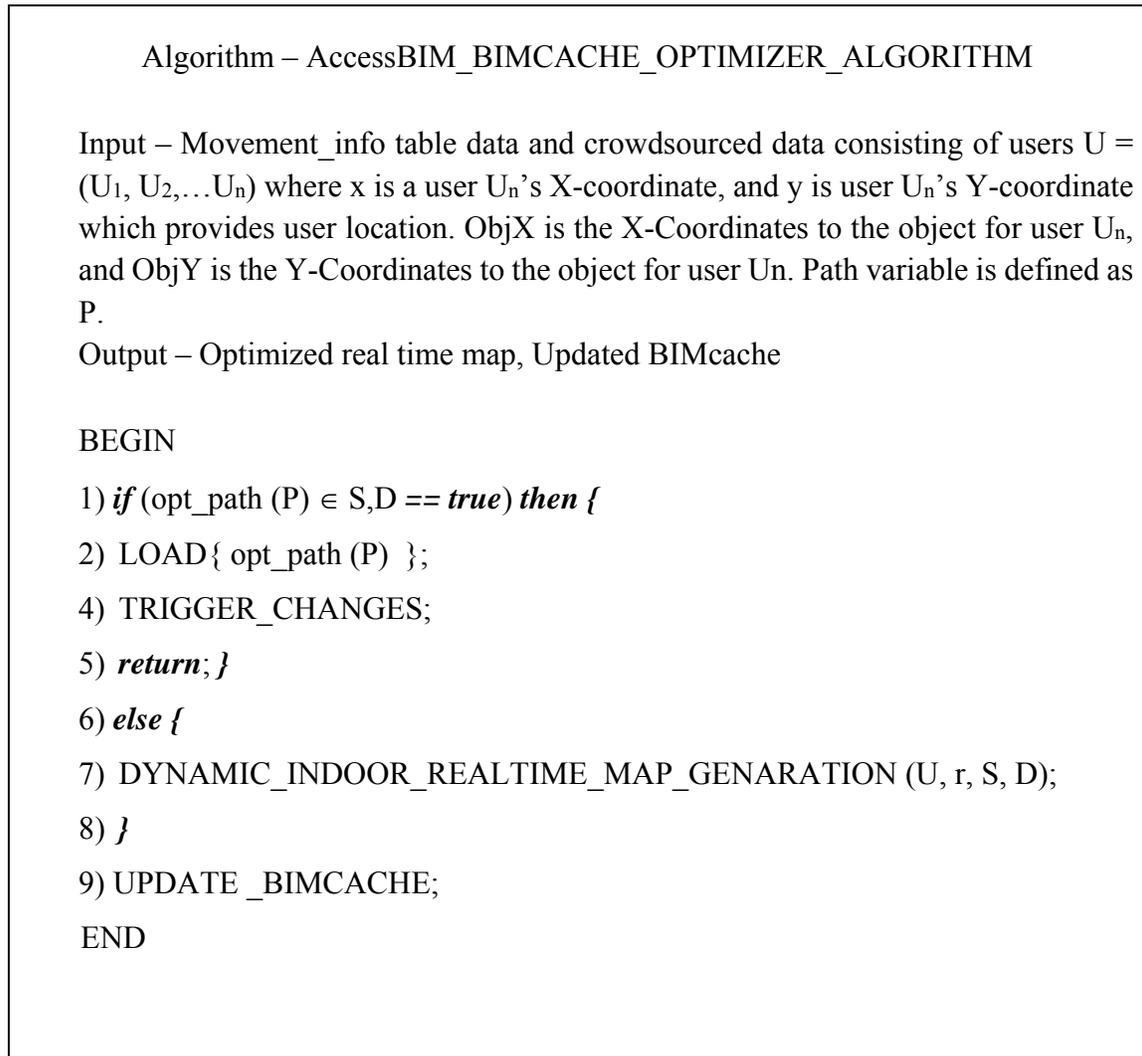


Figure 4.5: AccessBIM BIMcache Optimizer Algorithm

4.6 Algorithm for Crowdsourced Data Collection

Generating an indoor map requires collecting crowdsourced data via mobile devices. The accuracy of the data is the key element in this research, as the map is entirely based on the collected data. The collection of the data from crowdsourced environments should follow the steps mentioned in figure 4.6 to ensure accuracy of the data collected.

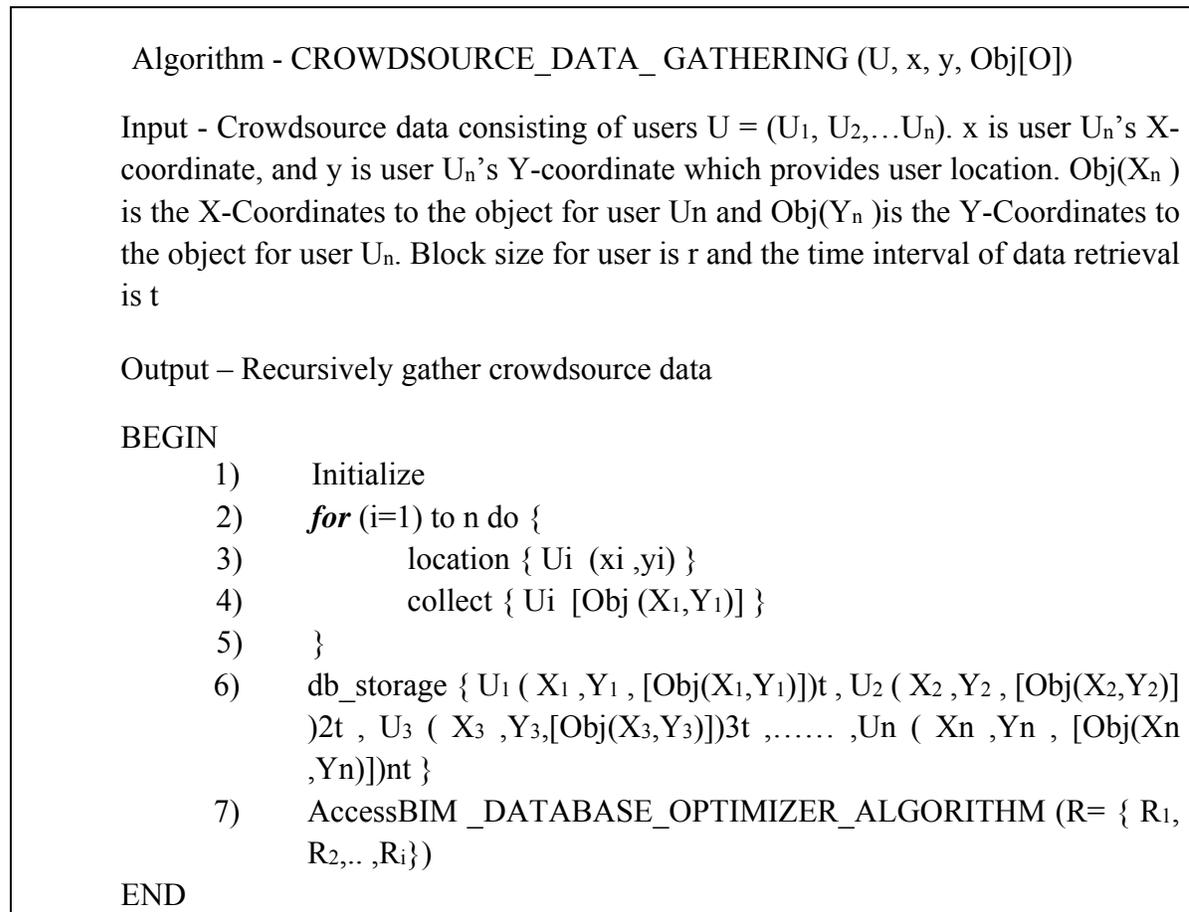


Figure 4.6: Crowdsourced Data Gathering

According to the CROWDSOURCE_DATA_GATHERING() algorithm, each user's location based on x, y coordinates and information on obstacles (x, y coordinates) within the environment of the user is gathered periodically (t). The gathered data is sent to the AccessBIM database for storage. At the end of each data collection session, the database is optimized using the AccessBIM_Database_Optimizer_Algorithm() that utilizes the user block size to define the user space in the environment.

4.7 Chapter Conclusion

This chapter introduced four algorithms developed by the author that facilitates map generation, data collection, storage and database optimization. The real-time map generation algorithm is used to generate the optimal path for navigation while the database optimization algorithm applies stored procedures and query rewriting to speed up data creation, insertion, updating and deletion. The algorithm for BIMcache optimization reduces the time and cost of searching data by examining the database for previously navigated paths while Crowdsourced data collection algorithm facilitates the generation of an accurate map through the identification of x, y coordinates.

The main objective of the above mentioned algorithms is to obtain a map with the support of a simulated environment and an optimized AccessBIM database. The process stages of each algorithm were followed clearly throughout the map generation process to obtain the intended outputs.

All the outcomes and discussions in chapter five are based on the algorithms introduced in this chapter. The above-mentioned algorithms have the capability of generating a map in real-time as described in section 3.4.

Chapter 5 – The Simulation Environment and Test cases

5.1 Chapter Overview

This chapter discusses how a real-time map is generated with the use of the algorithms presented in chapter 4. Section 5.2 gives a brief introduction to the simulation environment while section 5.3 describes five test cases that were used to assess the reliability of the simulation environment. Section 5.4 concludes the chapter with a brief discussion on how the use of the four algorithms and the simulation environment resulted in generating an accurate real-time map.

5.2 The Simulation Environment

This research was modelled on a simulation of a virtual environment that is similar to the test area which is the 7th floor at Sri Lanka Institute of Information Technology. As discussed in chapter 3, simulated data behaves similarly to actual data [19, 20]. The use of a computer simulation reduces the cost of building a controlled environment apart from reducing the time associated in creating a new system. The data was collected via a simulator to ensure the strength of the AccessBIM database and identify bugs and performance capabilities prior to the actual implementation.

A series of test cases described in section 5.3 were generated and tested using the simulation engine to justify the accuracy of the generated real-time map. In each of the test cases, the real-time map generated through the simulation was compared to the output expected by the author before the tests were carried out. The real-time map generated via the simulation environment and the AutoCAD floor plan on which the test cases were based were compared to determine the accuracy of the AccessBIM framework in a real-world scenario.

5.3 Test Scenarios

The AutoCAD floor plan is shown in figure 5.1. Based on the floor plan, a set of scenarios were created to check the accuracy of real-time map generation.

According to Figure 5.1 the 7th floor of Sri Lanka Institute of Information Technology is partitioned as; C – Cabin, D – Door ,E- Entrance ,EL –Elevators, L-Labs, W-Wash room,L1-

Lab 701, L2-Lab 702, L3-Multimedia Lab, R1-Assistance Room, R2-Maintenance Room, R3-Server Room, R4-Research Room and R5-Air Condition Control Room.

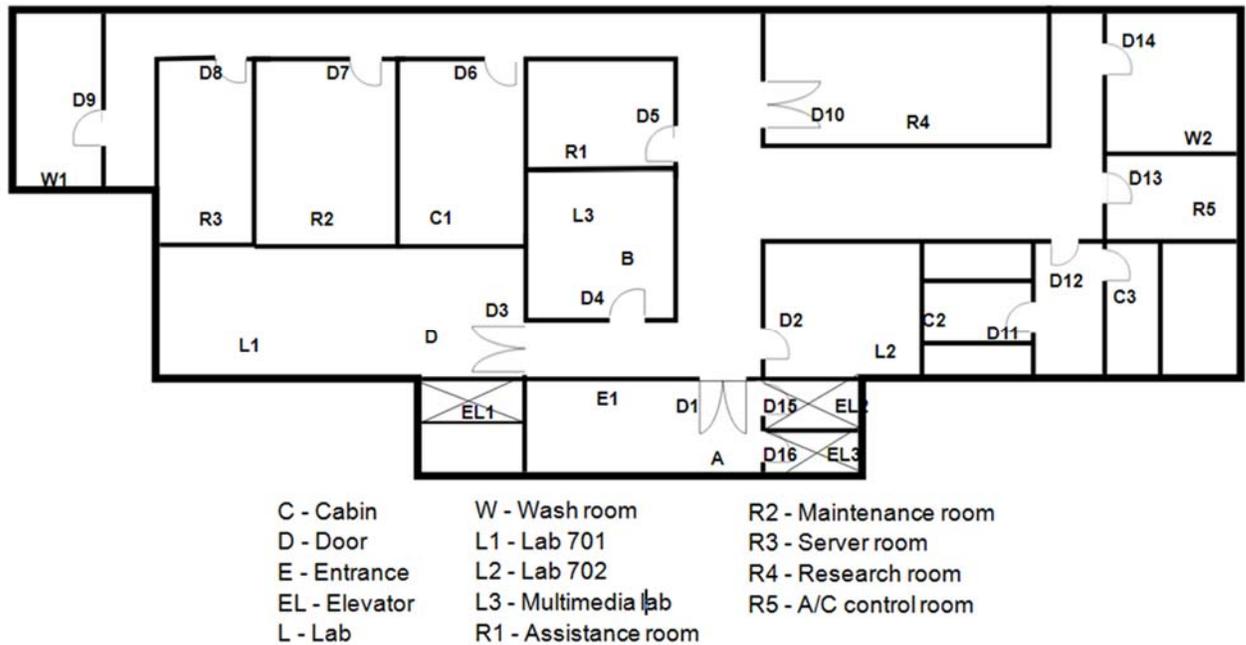


Figure 5.1: Floor plan of simulated environment (7th floor of Sri Lanka Institute of Information Technology)

In the framework described in figure 3.2, data was collected via an Application Programming Interface (API) and stored inside the AccessBIM database with the aid of the four algorithms discussed in chapter 4. The use of these algorithms ensured that the real-time map generated through the data collected from the simulation environment was similar to the map created through AutoCAD. Hence, the floor plan drawn using AutoCAD can be used to check the accuracy of the real-time map generated.

A series of test cases described in section 5.3 were generated and tested using the simulation engine to justify the accuracy of the generated real-time map. In each of the test cases, the real-time map generated through the simulation was compared to the output expected by the author before the tests were carried out. The real-time map generated via the simulation environment and the AutoCAD floor plan on which the test cases were based were compared to determine the accuracy of the AccessBIM framework in a real-world scenario.

The reasons for using a virtual environment (computer simulation) was to reduce the cost of building a controlled environment, time associated in creating a new system and the complexity in data collection. Furthermore, faster data retrieval and the detection of problems, bugs and difficulties in map generation were the major benefits of implementing a computer simulation before the actual implementation. The author used a test simulation engine which had the capability to collect and submit environmental data while a vision impaired individual moves within the simulated floor. The simulator consisted of functionalities such as the ability to provide localization information in the form of X, Y coordinates together with the distance (R) and direction (Θ).

Five test cases were generated and tested using the simulation engine to justify the accuracy of the real-time map generated of the 7th floor of Sri Lanka Institute of Information Technology. Further analysis on the test cases enabled the author to identify characteristics of the spatial environment that can effectively impact an individual. Each of these scenarios have been implemented separately as they have no relationship with each other. For instance, test case 1 examines the user's path from the main entrance to Lab 702 while test case 2 examines a user navigating from the Lab 701 to the multimedia lab with a change in the location of the obstacle. Thus, there would be no impact on the test results if the order of the test cases were reversed.

The five test cases given below demonstrate how user movements facilitate in generating the real-time map through the collection of crowdsourced data in each phase.

a) Test case 1: This test case examines the user's path from the main entrance (A) to Lab 702 (B) in order to justify the map generated through the data collected via the simulation environment with the aid of the algorithms discussed in chapter 4.

Scenario: A user navigates to a nearby destination (From the main entrance of the 7th floor to lab 702)

URL of video for test case 01 – <https://youtu.be/bLLgo1CUNrU> [71]

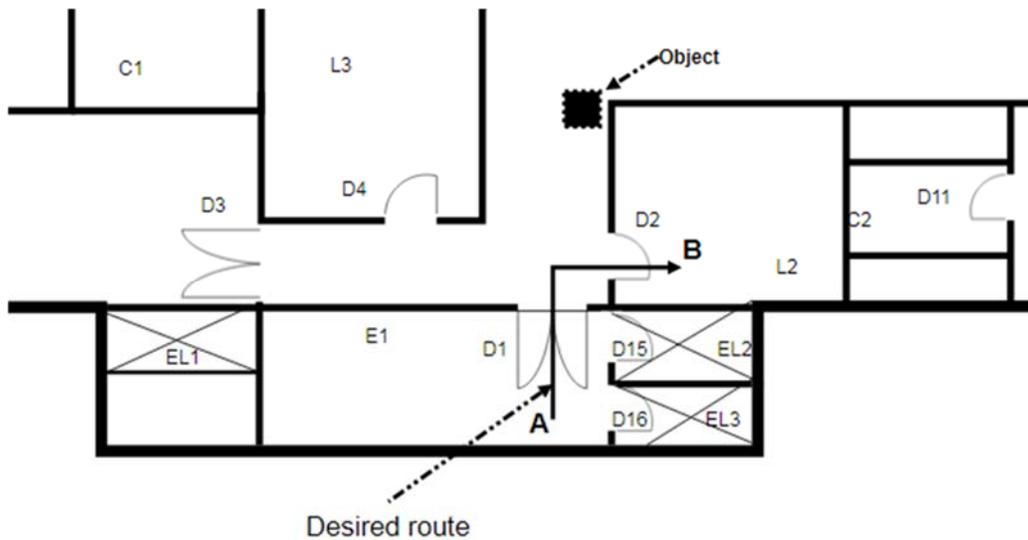


Figure 5.2: Desired navigation plan for scenario 1

Expected output:

Figure 5.2 depicts a portion of the 7th floor plan with the desired source and destination that the user wishes to reach. The doors are illustrated in green and the desired path of navigation in blue. It is expected that the user will pass door number 1 and 2 to reach lab 702. The map generated with the aid of the algorithms discussed in chapter 4 would include both doors, nearby walls, obstacles and the navigation path.

Database status (before navigation):

The structure of the AccessBIM database before navigation is observed as given below:

- Wall information, door information and object information tables contain X, Y coordinates of walls, doors and the current position of obstacles. (Figure 5.3, 5.4 and 5.5)
- Path history table contains details of source and destination available for map generation

- Movement table is empty

Data Output							
	Explain	Messages	History				
<input type="checkbox"/>	w_id [PK] inte...	f_id_fk_w [PK] inte...	b_id_fk_w [PK] inte...	starting_x real	starting_y real	ending_x real	ending_y real
<input type="checkbox"/>	1	7	1	0	0	796	0
<input type="checkbox"/>	92	7	1	0	0	0	143
<input type="checkbox"/>	93	7	1	0	142	107	142
<input type="checkbox"/>	94	7	1	73	0	73	55
<input type="checkbox"/>	95	7	1	73	99	73	142
<input type="checkbox"/>	96	7	1	107	54	133	54
<input type="checkbox"/>	97	7	1	167	54	196	54
<input type="checkbox"/>	98	7	1	240	54	268	54
<input type="checkbox"/>	99	7	1	306	54	391	54
<input type="checkbox"/>	100	7	1	107	183	306	183
<input type="checkbox"/>	101	7	1	306	142	391	142
<input type="checkbox"/>	102	7	1	306	221	351	221
<input type="checkbox"/>	103	7	1	106	286	393	286
<input type="checkbox"/>	104	7	1	468	286	800	286

Figure 5.3: Wall_info table

Data Output								
	Explain	Messages	History					
<input type="checkbox"/>	d_id [PK] inte...	f_id_fk_d [PK] inte...	b_id_fk_d [PK] inte...	starting_x real	starting_y real	ending_x real	ending_y real	isopen integer
<input type="checkbox"/>	1	7	1	393	287	473	287	1
<input type="checkbox"/>	2	7	1	305	223	305	287	1
<input type="checkbox"/>	3	7	1	351	221	392	221	1
<input type="checkbox"/>	4	7	1	390	103	390	142	1
<input type="checkbox"/>	5	7	1	268	54	306	54	2
<input type="checkbox"/>	6	7	1	196	54	240	54	1
<input type="checkbox"/>	7	7	1	133	54	167	54	1
<input type="checkbox"/>	8	7	1	73	55	73	99	1
<input type="checkbox"/>	9	7	1	468	70	468	123	1
<input type="checkbox"/>	10	7	1	718	20	718	58	1
<input type="checkbox"/>	11	7	1	718	135	718	175	1
<input type="checkbox"/>	12	7	1	670	184	709	184	1
<input type="checkbox"/>	13	7	1	669	217	669	256	1
<input type="checkbox"/>	14	7	1	468	236	468	283	1
<input type="checkbox"/>	15	7	1	468	298	468	329	1

Figure 5.4: Door_info table

Data Output Explain Messages History								
<input type="checkbox"/>	object_id [PK] inte...	lbl_id_fk integer	f_id_fk_o integer	b_id_fk_o integer	image_d... characte...	category characte...	x_coordi... real	y_coordi... real
<input type="checkbox"/>	1		1	1	chair	chair	25	89
<input type="checkbox"/>	2		1	1	Printr	printer	25	89
<input type="checkbox"/>	4		7	1	printer	printer	384	228

Figure 5.5: Object_info table

Image of the simulator (before navigation):

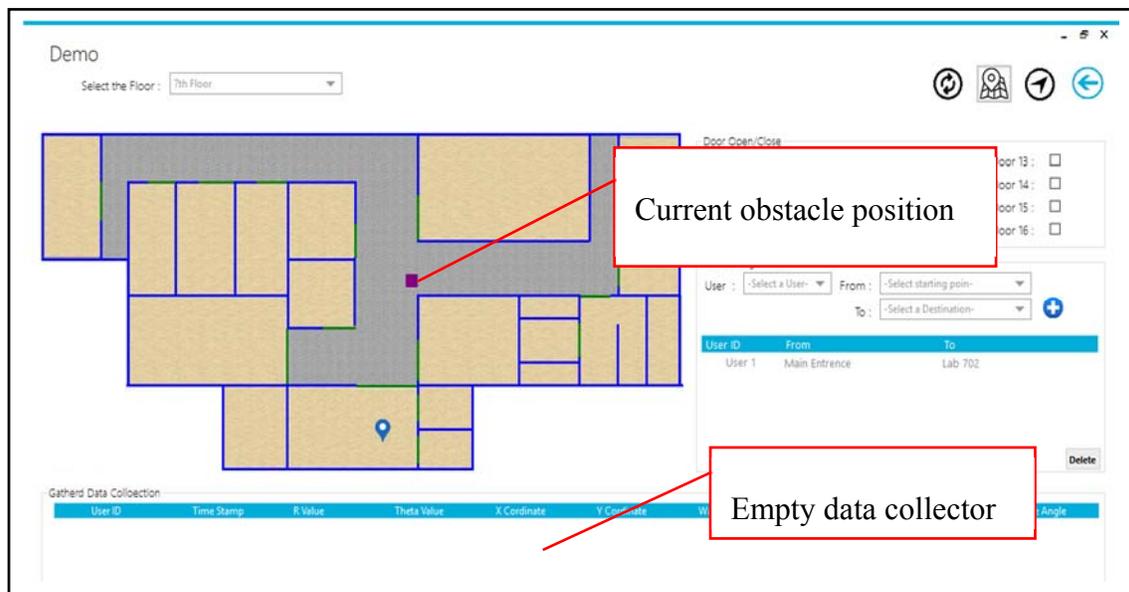


Figure 5.6: Simulator with object positioning

Database status after navigation (Figure 5.7):

- Wall, door and object information tables remain the same
- Path history table will be updated by adding source and destination
- Movement table will change with the navigation details

Data Output Explain Messages History										
<input type="checkbox"/>	moveme... [PK] inte...	object_i... integer	path_his... integer	walking_... real	angel real	direction characte...	time_sta... timesta...	x_coordi... real	y_coordi... real	z_coordi... real
<input type="checkbox"/>	83637	1	1024	0.8	0	North	2018-04-...	416	319	
<input type="checkbox"/>	83638	1	1026	0.8	90	East	2018-04-...	35	65	
<input type="checkbox"/>	83639	1	1022	0.8	90	East	2018-04-...	35	65	
<input type="checkbox"/>	83640	1	1025	0.8	90	East	2018-04-...	250	240	
<input type="checkbox"/>	83641	1	1023	0.8	0	North	2018-04-...	416	319	
<input type="checkbox"/>	83642	1	1022	1.6	0	East	2018-04-...	42	65	
<input type="checkbox"/>	83643	1	1026	1.6	0	East	2018-04-...	42	65	
<input type="checkbox"/>	83644	1	1025	1.6	0	East	2018-04-...	257	240	
<input type="checkbox"/>	83645	1	1023	1.6	0	North	2018-04-...	416	312	
<input type="checkbox"/>	83646	1	1024	1.6	0	North	2018-04-...	416	312	
<input type="checkbox"/>	83647	1	1022	2.4	0	East	2018-04-...	49	65	
<input type="checkbox"/>	83648	1	1026	2.4	0	East	2018-04-...	49	65	
<input type="checkbox"/>	83649	1	1025	2.4	0	East	2018-04-...	264	240	
<input type="checkbox"/>	83650	1	1023	2.4	0	North	2018-04-...	416	305	

Figure 5.7: Update movement_info table

The files in the movement information table are simultaneously updated while the user navigates as shown in figure 5.6.

Test Result:

The Real-time map generated for the given source and destination of the given test case scenario after successful navigation is shown on figure 5.8. It includes:

- Door 1 and 2
- Nearby walls
- Obstacle location as shown in figure 5.6. The location of the obstacles remain the same since the user did not navigate across the obstacle. Hence, the database does not provide any change in information relevant to the obstacle.
- Navigation Route to justify how the user reached from A to B as shown in test case 1.

When another user requests assistance to navigate to the same destination from the same source, the previously generated map would be loaded to their mobile device via BIMcache.

BIMcache is a caching system used to speedup map rendering by reducing the number of times the same data is accessed through the database as discussed under section 3.3.2

The output of test case 1 (figure 5.8) is the real-time map generated through the data collected via the simulation environment with the aid of the algorithms discussed in chapter 4. This real-time map matches with the output expected by the author before the simulation is carried out.

The real-time map generated in each of the five test cases, signify the doors by a blue line while a thick black line represent the walls. The dotted lines represent the path taken by the user to reach his desired destination.

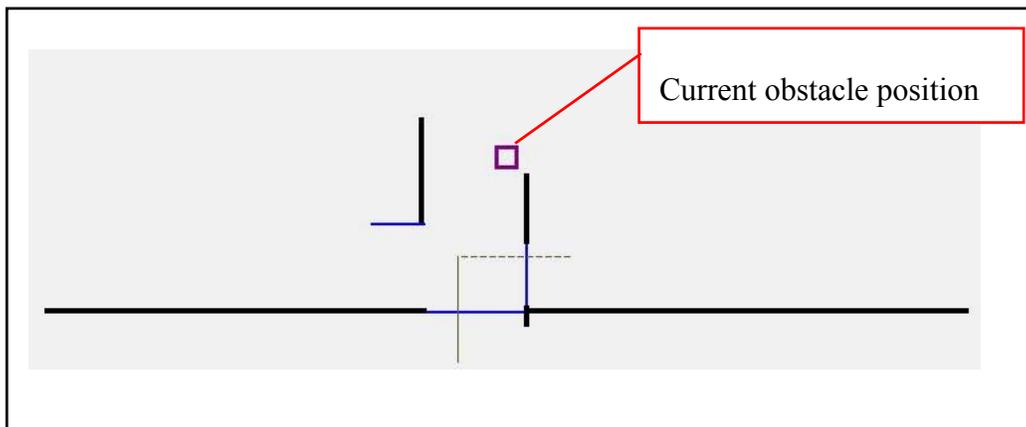


Figure 5.8: Generated real-time map according to navigation

b) Test case 2: This test case demonstrates how the algorithms discussed in chapter 4 identifies the obstacles and changes the movement based on the location of the obstacles and how the map is updated in real-time with the expected changes.

Scenario: A user navigates to a nearby destination with a change in the location of the obstacle. (From the Lab 701of the 7th floor (A) to the multimedia lab (B) after changing the obstacle's previous position as shown in figure 5.9)

URL of video for test case 02 - https://youtu.be/-7t_svmBuXw [72]

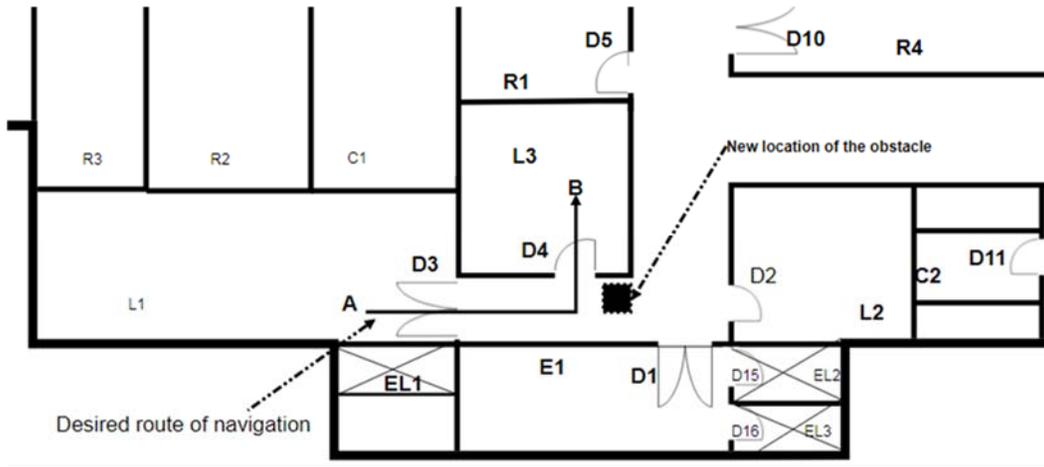


Figure 5.9: Desired navigation plan for scenario 2

Expected output:

The user will pass door number 3, 4 and the obstacle. The map that is generated in real time will include both doors, walls passed in test case 1 and 2, the new location of the obstacle and the navigation path. Since test case 2 is carried out after test case 1, movement details pertaining to test case 1 will be already stored in the AccessBIM database.

Database status (before navigation):

The structure of the AccessBIM database before test case 2 is observed as given below:

- Wall information, door information and object information tables contain X, Y coordinates of walls, doors and the current position of obstacles.
- Path history table contains details of source and destination available for map generation.
- The movement table contains data of test case 1; figure 5.10 demonstrates the movement information table after test case 1 while the status of the simulator after changing the obstacle's location is illustrated in figure 5.11.

	moveme... [PK] inte...	object_i... integer	path_his... integer	walking_... real	angel real	direction characte...	time_sta... timesta...	x_coordi... real	y_coordi... real	z_coordi... real
<input type="checkbox"/>	83637	1	1024	0.8	0	North	2018-04-...	416	319	
<input type="checkbox"/>	83638	1	1026	0.8	90	East	2018-04-...	35	65	
<input type="checkbox"/>	83639	1	1022	0.8	90	East	2018-04-...	35	65	
<input type="checkbox"/>	83640	1	1025	0.8	90	East	2018-04-...	250	240	
<input type="checkbox"/>	83641	1	1023	0.8	0	North	2018-04-...	416	319	
<input type="checkbox"/>	83642	1	1022	1.6	0	East	2018-04-...	42	65	
<input type="checkbox"/>	83643	1	1026	1.6	0	East	2018-04-...	42	65	
<input type="checkbox"/>	83644	1	1025	1.6	0	East	2018-04-...	257	240	
<input type="checkbox"/>	83645	1	1023	1.6	0	North	2018-04-...	416	312	
<input type="checkbox"/>	83646	1	1024	1.6	0	North	2018-04-...	416	312	
<input type="checkbox"/>	83647	1	1022	2.4	0	East	2018-04-...	49	65	
<input type="checkbox"/>	83648	1	1026	2.4	0	East	2018-04-...	49	65	
<input type="checkbox"/>	83649	1	1025	2.4	0	East	2018-04-...	264	240	
<input type="checkbox"/>	83650	1	1023	2.4	0	North	2018-04-...	416	305	

Figure 5.10: Movement_info Table after test case 1

Image of the simulator (before navigation):

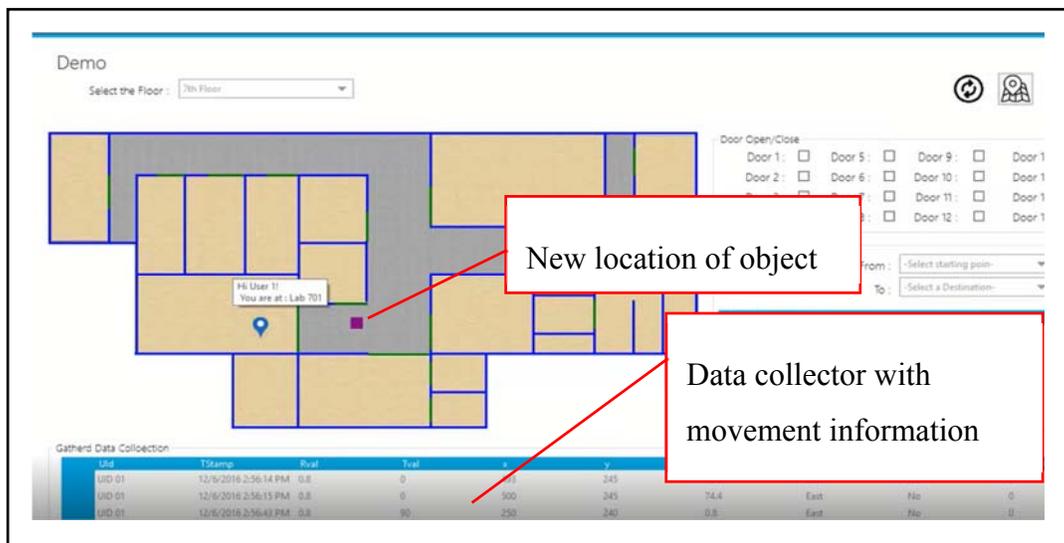


Figure 5.11: Simulator after changing object's location

Database status (after navigation):

- Wall_info, Door_info table remain the same
- Object table is updated with new X, Y coordinates of the obstacle

- Movement table is changed with the current navigation details (Movement_info table contains foreign keys for door_info and object_info tables to identify updates)

Test result:

-The map is updated with,

- Door 1 & 2 from test case 1
- Door 3 & 4 from test case 2
- All walls that were passed
- The new location of the object
- Navigation route; after successful navigation within the simulator.

The output of test case 2 (figure 5.12) is the real-time map generated through the data collected via the simulation environment with the aid of the algorithms discussed in chapter 4. This real-time map matches with the output expected by the author before the simulation is carried out hence, it justifies the accuracy of the simulation and the four algorithms.

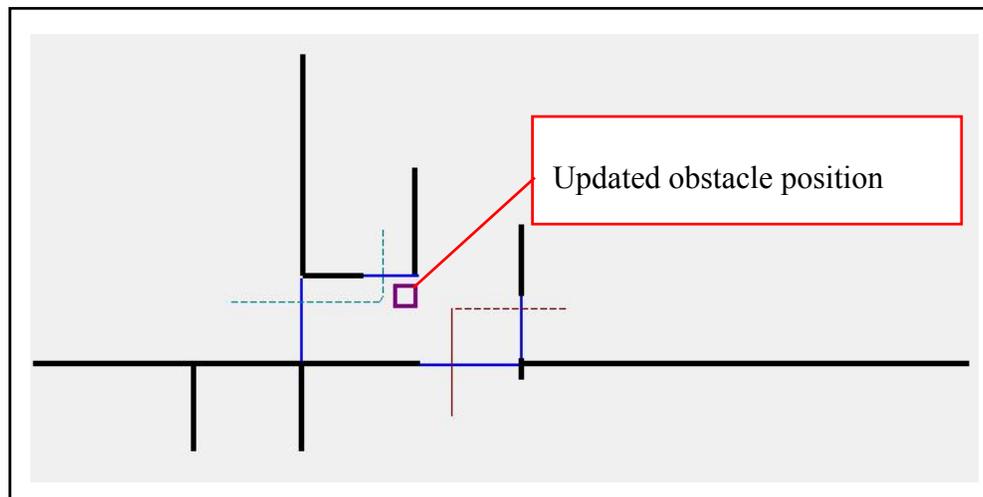


Figure 5.12: Updated real-time map with new object location

c) *Test case 3: This test case is carried out to demonstrate how the algorithms discussed in chapter 4 facilitates identification of the current position of the obstacle, the remote locations from the source and how the map is updated in real-time with the expected changes.*

Scenario: A user navigates to a remote destination with replacement of the previous obstacle. (From the main entrance of the 7th floor (A) to cabin 1 (B) located far from the main entrance. Figure 5.13 illustrates a part of the floor plan of the 7th floor of Sri Lanka Institute of Information Technology.

URL of video for test case 03 - <https://youtu.be/2U6BF4Wcxm0> [73]

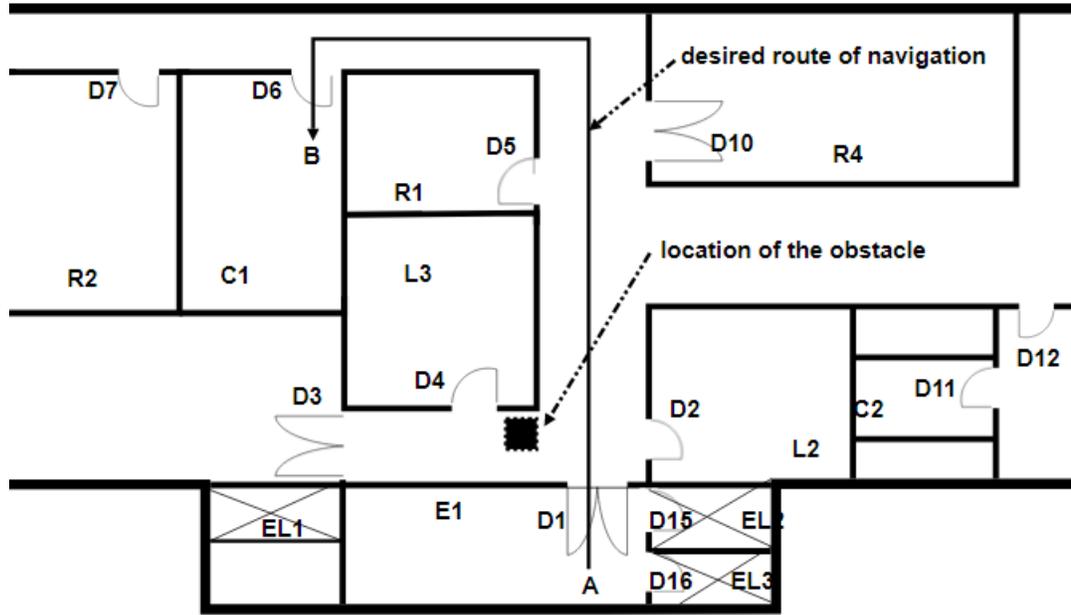


Figure 5.13: Desired navigation plan for scenario 3

Expected output:

It is expected that the user will pass door number 1, 6 and the obstacle. The real-time map will be generated with all the doors passed including door 1 and 6, walls passed in test case 2 and 3, new location of the obstacle and the navigation path. Since test case 3 is carried out after test case 1 and 2, movement details pertaining to test case 1 and 2 will be already stored in the AccessBIM database.

Database status (before navigation):

The structure of the AccessBIM database before test case 3 is observed as given below:

- Wall information, door information and object information tables contain X, Y coordinates of the walls, doors and the current position of the obstacles.
- Path history table contains the details of source and destination available for map generation.
- The movement table contains data of test case 1 and 2

Database status (after navigation):

- Wall_info, Door_info, Object_info tables do not change
- Path history table is updated by the addition of new source and destination details
- Movement table is updated with the new path history id

Test result:

The map is updated with the new source and destination after successful navigation with the current object positioning, by filtering data in the movement_info table. The output of test case 3 (figure 5.14) is the real-time map generated through the data collected via the simulation environment with the aid of the algorithms discussed in chapter 4. This real-time map matches with the output expected by the author before the test is carried out.

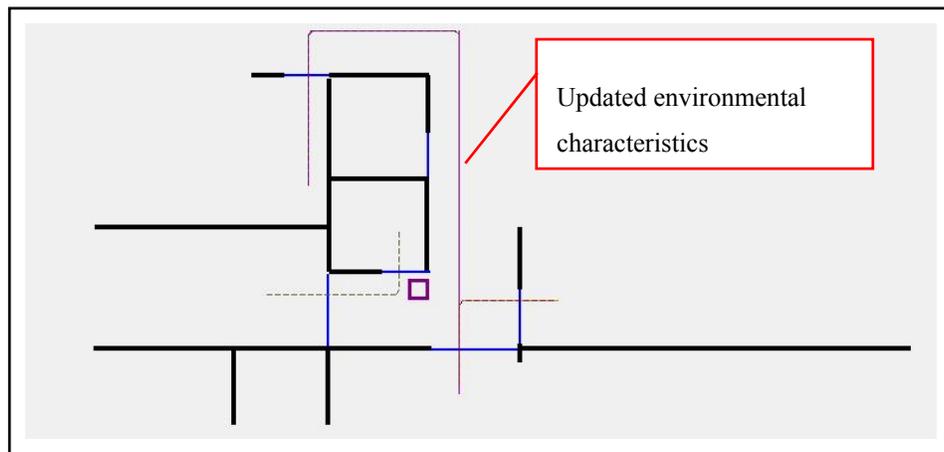


Figure 5.14: Updated real-time map with new environmental characteristics

d) Test case 4: The aim of this test case is to demonstrate how the algorithms facilitate multiple user navigation and how the map is updated in real-time with the expected changes

Scenario: Multiple users are trying to navigate to multiple destinations with the replacement of the previous obstacle when all doors are open.

URL of video for test case 04 - <https://youtu.be/MiYx7STJuRQ> [74]

- User1 navigates to washroom 2 (B) from the main entrance (A) via door 1 & 14,
- User 2 navigates to cabin 2 (C) from the main entrance (A) via door 1, 11 and 12,
- User 3 navigates to the multimedia lab (E) from lab 701 (D) via door 3 and 4,
- User 4 navigates to cabin 1 (G) from the washroom 1 (F) via door 9 and door 6.

Expected output:

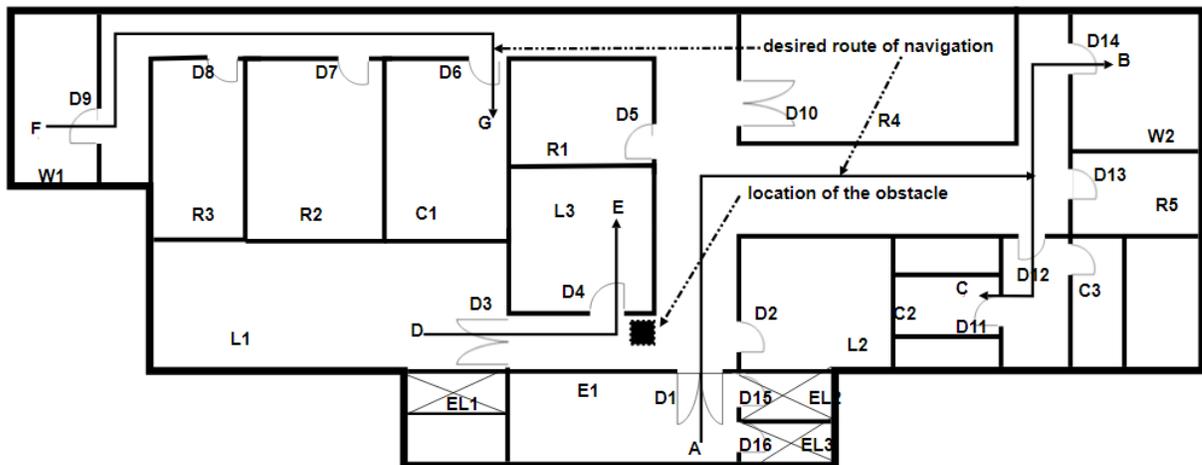


Figure 5.15: Desired navigation plan for scenario 4

It is expected that the real-time map will be generated with door number 1, 3, 4, 6, 11, 12, and 14, in addition to all the walls passed by the users in previous and current test cases and the new location of the obstacle.

Database status (before navigation):

The structure of the AccessBIM database before navigation is observed as given below:

- Wall information, door information and object information tables contain X, Y coordinates of walls, doors and the current position of the obstacles.
- Path history table contains details of source and destination available for map generation.
- Movement table contains data of test case 1, 2 and 3

Image of the simulator (when navigating):

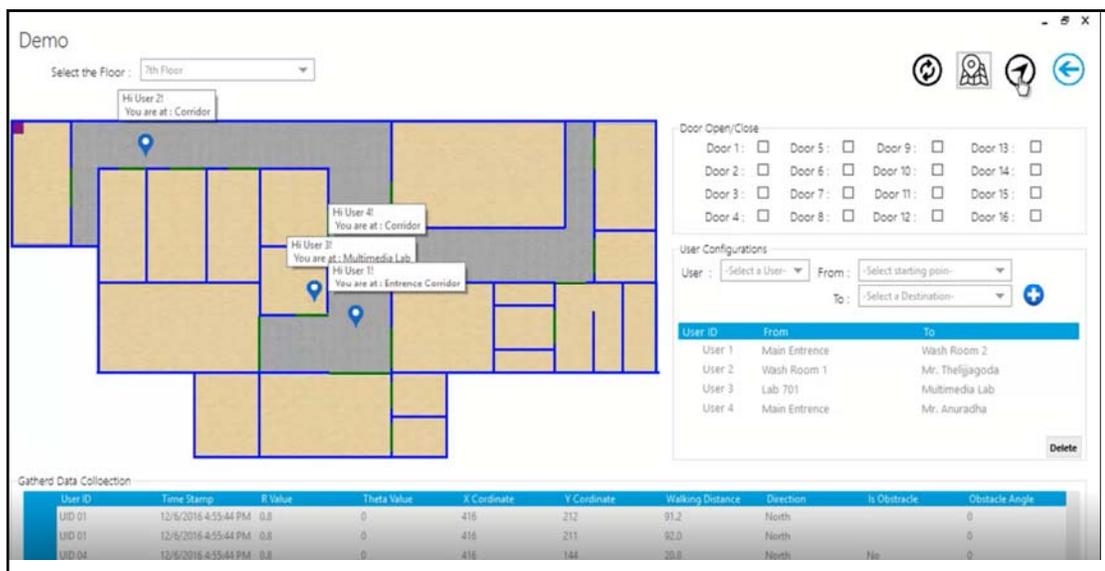


Figure 5.16: Multiple user navigations at once

Database status (after navigation):

- Wall_info, Door_info and obstacle_info tables do not change
- Movement table changes with the current navigation details for all users simultaneously

Test result:

After successful navigation, the map is updated with all the routes that are currently available and the new positions of the objects, by filtering the data in movement_info table.

The map contains:

- Door 1, 3, 4, 6, 11, 12, and 14

- Walls passed by the users in previous test cases
- Walls passed by the users in the current test case
- Current location of the obstacle

The output of test case 4 (figure 5.17) is the real-time map generated through the data collected via the simulation environment with the aid of the algorithms discussed in chapter 4. This real-time map matches with the output expected by the author before test case 4 is carried out.

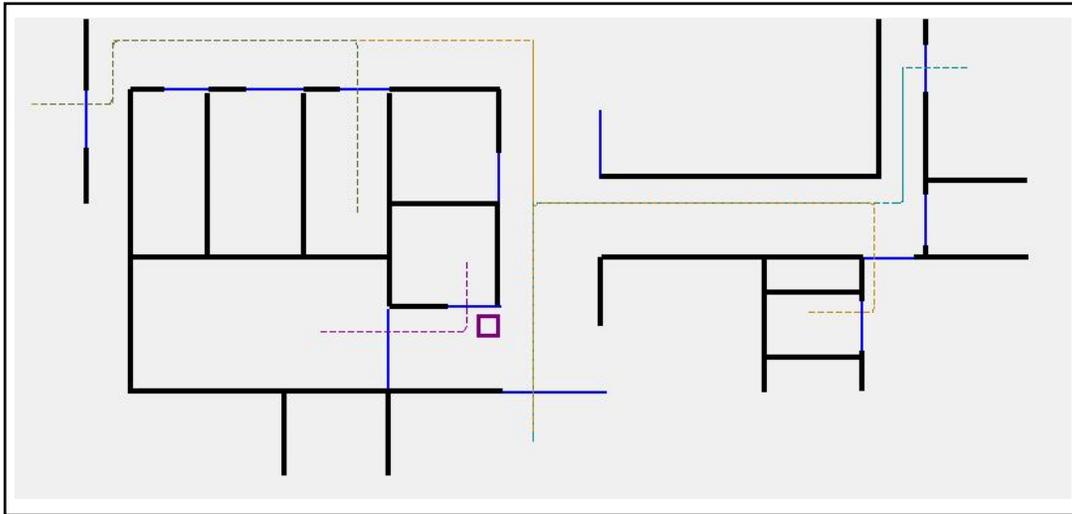


Figure 5.17: Updated map with multiple user navigations (Map is approximately complete)

The real-time map generated as the output of test case 4 is very much similar to the floor plan of the 7th floor drawn using AutoCAD. (Figure 5.1) This indicates that when more users navigate within an environment more and more information is collected which would result in generating a real-time map that is accurate.

e) Test case 5: The aim of test case 5 is to demonstrate how the algorithms identify the status of the doors while navigating and update the database with that information

Scenario: A user navigates to a previously entered room when the door is closed.
(From the main entrance to the multimedia room when the door is closed)

URL of video for test case 05 - <https://youtu.be/qunEWHBpKR4> [75]

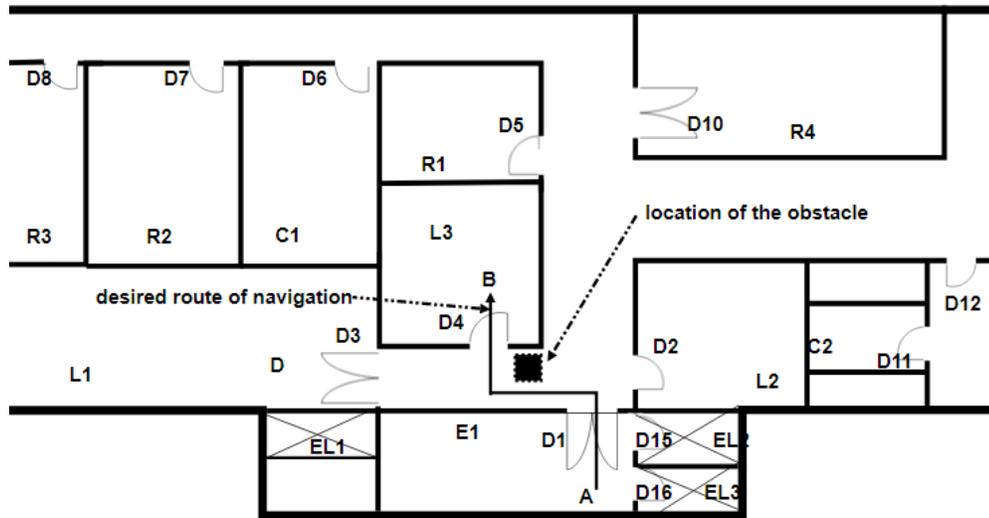


Figure 5.18: Desired navigation plan for test case 5

Expected output:

It is expected that the real-time map updates the status of door 4 as closed and prevents the user from entering the multimedia lab. The real-time map should contain all the doors and walls the user meets while navigating and the current position of the obstacle. Soon after this, if a new user tries to enter the multimedia lab, the loaded real-time map will show that the door is closed indicating that it is impossible to enter the lab.

Database status (before navigation):

The structure of the AccessBIM database before navigation is observed as given below:

- Wall information, door information and object information tables contain X, Y coordinates of walls, doors and the current position of obstacles.
- Path history table contains details of source and destination available for map generation.
- Movement table contains data of the previous test cases

Image of the simulator (when navigating):

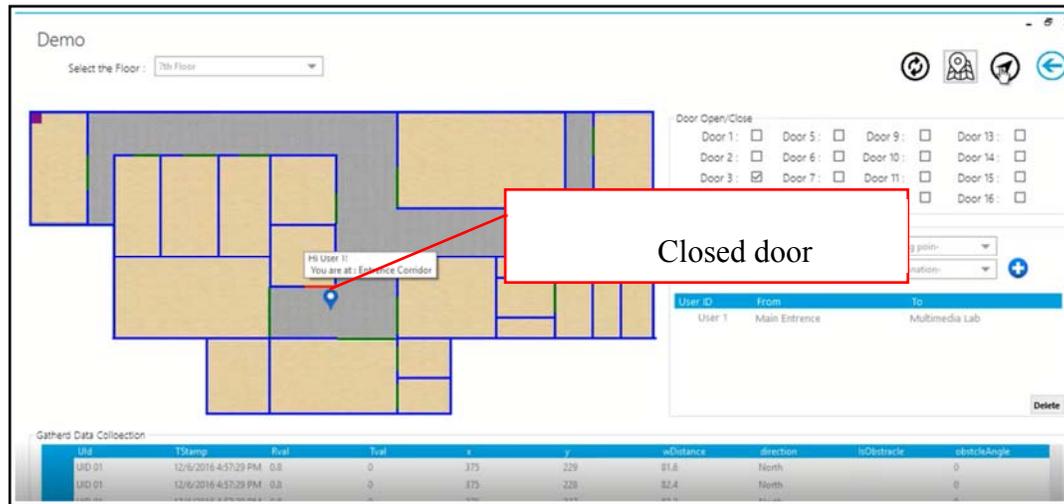


Figure 5.19: Door closed in simulator

Database status (after navigation):

- Wall_info table does not change
- Door_info table is updated with the new door status as “door open -> no” for the selected door (default door status is “door open -> yes”)
- Movement_info table is replaced with the current navigation details

Test result:

The real-time map is updated with the current status of the door after each successful navigation, the current position of the objects and the status of the door by filtering data in the movement_info table

The user is unable to enter a room which he previously entered as the simulation identifies the closed door as an obstacle and prevents the user from moving further.

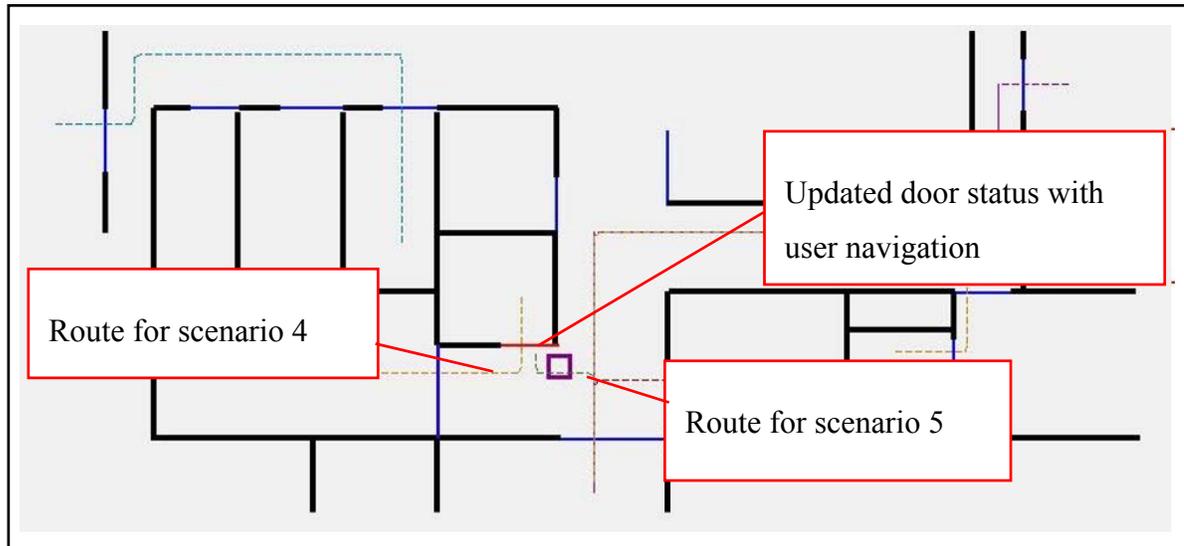


Figure 5.20: Updated map with current door status and routes

5.4 Chapter conclusion

The AccessBIM framework collects and stores simulated crowdsourced data with the aid of the four algorithms discussed in chapter 4. The algorithms for crowdsourced data collection, database optimization, BIMcache optimization and indoor real-time map generation ensured that the real-time map generated through the data collected from the simulation environment was similar to the map drawn manually through AutoCAD.

Since the movement details of the previous test cases are already stored in the AccessBIM database, it can be observed that an increase in the number of users navigating to various locations would result in generating a real-time map that is accurate as more and more crowdsourced data is collected.

In each of the test cases, the real-time map generated through the simulation was in accordance with the output expected by the author before the tests were carried out. This justifies the accuracy of the simulation and the real-time map that was generated in each test case. However, a major issue encountered during the implementation of the test cases were the accumulation of records with the increasing number of users in the PostgreSQL database

which had to be manually cleared out. Continuous execution of the code due to the application of multiple users resulted in increased time consumption in running the simulation. In addition access was limited to internal simulation data due to issues faced in transferring data from the simulation to the database which was mitigated through the use of Dynamic Link Libraries (DLL). This was mainly due to the fact that the simulator did not support long, real or integer type data for which DLL was required to map different data types to ensure compatibility.

Furthermore, the default RID RIT file used in configuring the simulator environment required to be edited each time a new scenario was being tested. Similarly, the delivery of data was affected by the difference in network time which was later overcome by synchronizing a centralized network time server wherever possible. One other major issue faced in implementing the test cases were the training of users to identify and avoid obstacles. Thus, a Probabilistic Graphical Model (PGM) was used for extracting features and a generic geometric model was built to detect objects by coalescing edges and corners

In conclusion, it can be further stated that the use of the four algorithms ensured efficient and reliable data collection and accurate real-time map generation.

Chapter 6 – Evaluation and Discussion

6.1 Chapter Overview

This chapter evaluates the performance of the AccessBIM framework in generating a real-time indoor map through the use of database optimization techniques such as stored procedures, indexing and query rewriting. Section 6.2 evaluates the performance of the AccessBIM framework in terms of the database optimization techniques that were utilized in the research while section 6.3 compares the total time taken for real-time map generation. Section 6.4 compares the AccessBIM framework with a similar database optimization framework. The fundamental goal of the AccessBIM framework was to generate a real-time map to facilitate indoor navigation among the vision impaired individuals. Section 6.5 compares the AccessBIM framework with existing solutions while section 6.6 concludes the chapter with a brief discussion on how this goal was successfully achieved.

6.2 Evaluation of the AccessBIM framework

The AccessBIM model is a digital representation of the indoor building features which facilitates the exchange and interoperability of real-time information thereby assisting vision impaired individuals to independently access unfamiliar building indoor environments. The AccessBIM framework (Figure 3.2) collects simulated crowdsourced data through an API to generate a real-time map that would facilitate vision impaired indoor navigation. Since the indoor map is to be generated in real-time, the database must be optimized to process the queries in real-time.

Research on database optimization has been constantly evolving as the design of a database must cater to both the needs of the clients as well as the efficiency of the database processes. Database optimization immensely facilitates tasks that have critical timelines. The AccessBIM framework utilizes query optimization and indexing which have a significant influence on database optimization.

The AccessBIM framework facilitates in saving time as collecting data via crowdsourcing is much faster compared to collecting data from mobile robots. The use of the AccessBIM

framework also ensures less confusion as the map is being updated regularly with real-time changes of the indoor environment. Furthermore, the use of database optimization techniques speeds up the process of query execution enabling the map to be generated as and when requested by a user.

The AccessBIM database is capable of achieving the above-mentioned benefits due to faster query processing, lesser cost per query and efficient use of the database engine with less memory consumption. Indexing makes it convenient to search the required data from large data sets while query optimization facilitates the identification of the best query for retrieving data using minimal resources thus enabling high performance of the system via faster processing and lesser database cost thereby increasing the performance of the real-time map generation layer. This leads to efficient usage of the database engine with less memory consumption as the database is generated without data duplication. Thus, query optimization enables faster data insertion, updation, deletion and retrieval.

The AccessBIM framework utilizes an Object Relational Database (ORDB) to store spatial characteristics of indoor environments. The use of an ORDB provides numerous benefits over the use of traditional relational databases [76]. The need, to handle an enormous amount of IMU data together with the use of many stored procedures and indexes were one of the major reasons behind selecting an Object Relational Database despite the fact that it is complex and costly.

Consequently, the main advantage of using an ORDB for the AccessBIM framework is its reusability by providing the ability to store standard procedures (or functions) on the server, rather than coding them in each execution. For example, the application needs to call the function “check_path_availability” each time when the user selects source and destination. If the stored procedure for checking path availability is embedded in the server, it saves time in having to define them, each time the function is called.

Apart from reusability, the use of an ORDB has enabled the AccessBIM framework to increase its flexibility and functionality. ORDB's can be easily maintained with respect to relational databases and is easily extensible and reliable than traditional databases.

Furthermore, systems that use ORDB's have reduced response times in executing simple, user-defined queries compared to relational databases [76] which is the main reason behind implementing an ORDB for the AccessBIM framework despite its limited performance due to database cost.

6.3 Database optimization

Database optimization techniques such as stored procedures and indexing facilitate the AccessBIM framework to generate an effective real-time map. The ability to compile and reuse the execution plan to provide efficient performance boots is one of the major benefits that can be obtained in utilising stored procedures. After refining the schema it was identified that 32 stored procedures were required to eliminate the database overhead. Hence, 32 stored procedures were created to insert, delete, select, check and update data within the AccessBIM database. The use of stored procedures facilitated complex processing of several query statements by enabling input parameters and multiple output parameters rather relying on SQL queries. Given below are few factors that need to be considered in generating a real-time map.

- Check whether the requested path is available in BIMcache.
- Get details from the path_history_info table regarding the starting and ending points of the route if the movement details are not available in the BIMcache.
- Insert movement information while the user navigates within the simulator.
- Select wall, door and object information from the database that is specific to the navigation information (Environmental data related to that particular navigation)
- Update door information table about door status (whether the doors are open or not).
- Update object information table with the current location of the obstacles/objects.

In order to generate an accurate indoor map, the above-mentioned factors should be strictly followed in every map generation instance. Hence, the author decided to add indexing strategies to the stored procedures as they have the capability of fulfilling the above-mentioned tasks.

6.3.1 Performance Enhancement with Stored Procedures and Indexing For Real-Time Indoor Map Generation

Robot based solutions are the most common practices available to generate a real-time map [40, 44]. However, robot based solutions neither utilize stored procedures nor indexing strategies to retrieve data for map generation. Hence, robot based solutions are unable to experience the benefits of using stored procedures for real-time map generation.

The benefits of stored procedures can be listed as follows:

- Reutilizing the execution plan to provide efficient performance boots
- Ability to collect multiple input parameters to generate the real-time map (UDF allows only one input parameter at once)
- Minimizing round trips between the database and application to save execution time by enfolding all SQL statements inside a function stored in PostgreSQL database server. (The AccessBIM framework utilizes a PostgreSQL database as its DBMS)

Since this research is based on crowdsourced data, the solution customizes the stored procedures and applies indexing for the ones that have the greatest impact on the query execution time. Out of the thirty two (32) stored procedures listed on Appendix F, indexing was applied to the (eight) 8 stored procedures listed below. The following stored procedures were indexed to increase the performance of the real-time indoor map generation.

- BIMcache()
- select_values_from_wall_info()
- select_values_from_door_info()
- update_door_info()
- select_starting_and_ending()
- update_object_info()
- get_routing_details_for_a_specific_day()
- select_object_info()

The main reason behind selecting the above-mentioned stored procedures is their ability to support major tasks that require to be completed in the map generation process. Each map generation instance utilizes the selected stored procedures with indexing together with the remaining 24 stored procedures without indexing, to generate maps that are accurate. The next section discusses the above-mentioned stored procedures in detail.

6.3.1.1 Retrieve details regarding previous navigations

Stored procedure name: BIMcache ()

When a user successfully navigates to a certain destination, his movement information will be added to BIMcache. Using that movement information, a real-time map could be generated. If another user requires to reach the same destination from the same source, the generated map will be automatically loaded to the user via the BIMcached data without allowing them to navigate using the simulator. Generating and loading the real-time map via BIMcache reduces the time of query execution and increases the performance. Two B-tree indexes were implemented on the BIMcache stored procedure for performance enhancement.

The flow of the BIMcache () stored procedure:

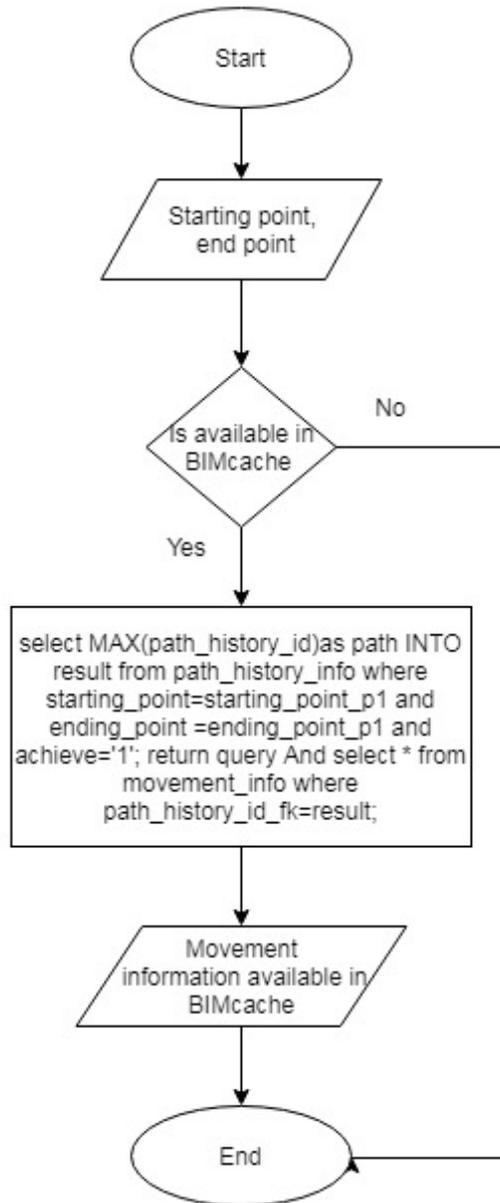


Figure 6.1: Flow of BIMcache () stored procedure

Index name: path_history_info_to_starting and path_history_info_to_ending

Implemented index:

//create index on starting point

CREATE INDEX path_history_info_to_starting ON path_history_info (starting_point ASC)

```
//create index on ending point
```

```
CREATE INDEX path_history_info_to_ending ON path_history_info (ending_point  
ASC)
```

Results of indexing

The test was conducted with three users and they had to navigate to a nearby location, a remote location and undergo an obstacle replacement. The total runtime of the BIMcache stored procedure was found before and after indexing was applied. The mean values were calculated for both scenarios after executing the stored procedure for 5 times. The mean runtime of the BIMcache stored procedure before indexing was 3.348 s whereas the mean runtime of the BIMcache stored procedure after indexing turned out to be 0.880 s.

Cost comparison for index on BIMcache () stored procedure

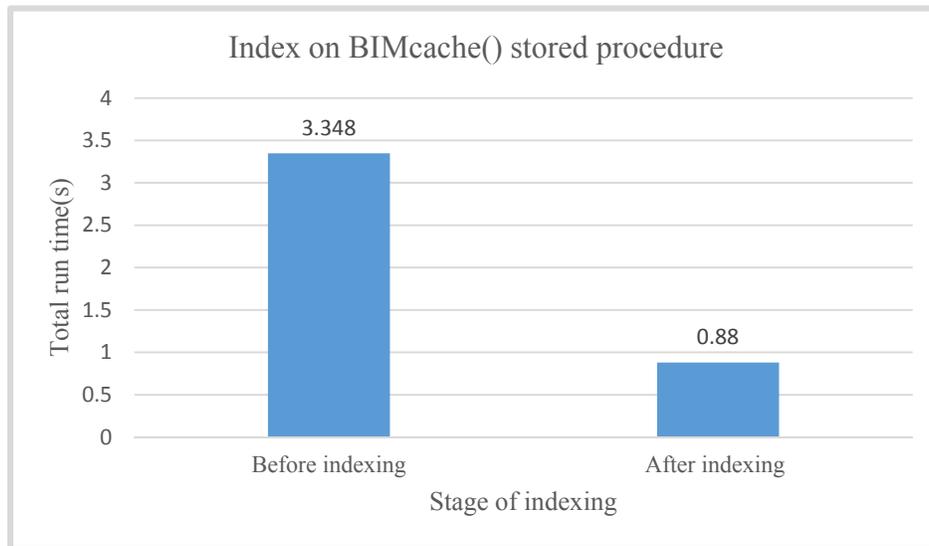


Figure 6.2: Cost comparison for stored procedure with indexing

The graph in figure 6.2 illustrates the reduction in the total runtime before and after indexing was applied on the BIMcache stored procedure.

6.3.1.2 Obtain path history information

Stored procedure: `select_starting_and_ending ()`

This stored procedure is used to find the path history id for a given source and destination from the path_history_info table where the end user successfully reaches the destination (achieve = 1). Path history id filters movement information for map generation from the movement_info table, which generates the real-time map. Due to the importance of obtaining the path history id, the performance of the stored procedure was increased by adding a partial index on it. Figure 6.3 illustrates the flow of select_starting_and_ending () stored procedure.

Flow of the select_starting_and_ending () stored procedure:

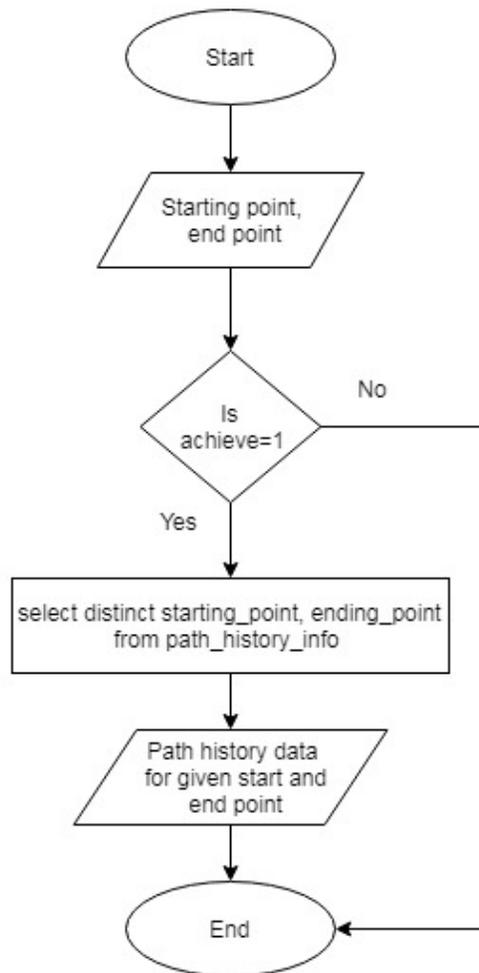


Figure 6.3: Flow of select_starting_and_ending () stored procedure

Index name: path_history_info_get_achieve

Implemented index:

```
//create index on successfully reached path history information

CREATE INDEX path_history_info_get_achieve ON path_history_info (achieve)
WHERE achieve = '1'
```

Results of indexing

The test was conducted with three users after allowing them to navigate to a nearby location, a remote location and undergo an obstacle replacement. The total runtime of the `select_starting_and_ending ()` stored procedure was found before and after indexing was applied. The mean values were calculated for both scenarios after executing the stored procedure for 5 times. The mean runtime of the `select_starting_and_ending ()` stored procedure before indexing was 0.789 s whereas the mean runtime of the `select_starting_and_ending ()` stored procedure after indexing turned out to be 0.175 s.

Cost comparison for index on `select_starting_and_ending ()` stored procedure

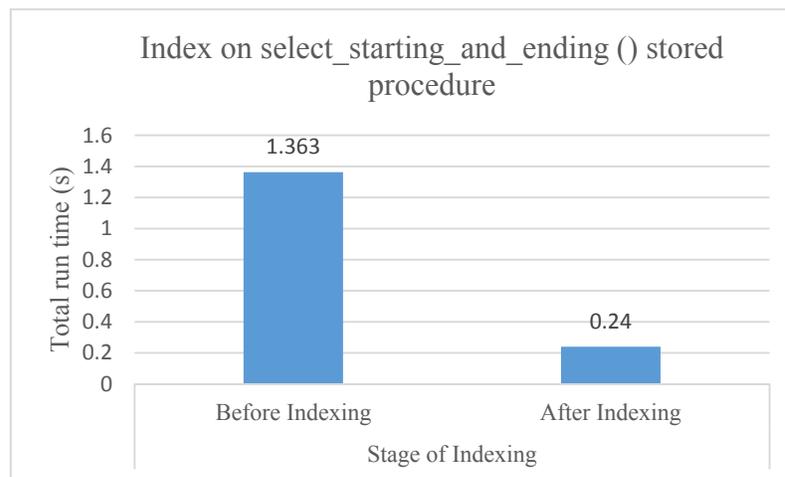


Figure 6.4: Cost comparison of `select_starting_and_ending ()` stored procedure

The graph in figure 6.4 illustrates the reduction in the total runtime before and after indexing was applied on the `select_starting_and_ending ()` stored procedure

6.3.1.3 Obtain wall information

Stored procedure: `select_values_from_wall_info ()`

After navigating within the simulator, environmental data should be collected based on the X, Y coordinates. The X, Y coordinates of the walls can be collected by obtaining the coordinates of the start and end points of the wall together with the coordinates in between. The walls passed by the users are the most important environmental data that can facilitate users in navigation. Hence, the “select_values_from_wall_info” stored procedure was created to filter out accurate wall information. Two B-tree indexes were implemented on this stored procedure to increase the efficiency of the data filtering process. Figure 6.5 illustrates the flow of select_values_from_wall_info () stored procedure

Flow of the select_values_from_wall_info () stored procedure:

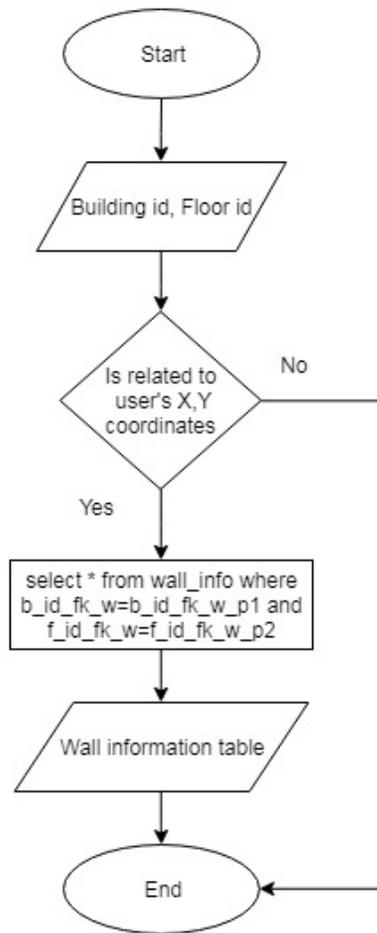


Figure 6.5: Flow of select_values_from_wall_info () stored procedure

Index names: wall_info_select_with_building_id

wall_info_select_with_floor_id

Implemented indexes:

```
//create index on building id of wall_info table to retrieve walls inside specific building
```

```
CREATE INDEX wall_info_select_with_building_id ON wall_info(b_id_fk_w ASC)
```

```
//create index on floor id of wall_info table to retrieve walls inside specific floor
```

```
CREATE INDEX wall_info_select_with_floor_id ON wall_info(f_id_fk_w ASC)
```

Results of indexing

The test was conducted with three users after allowing them to navigate to a nearby location, a remote location and undergo an obstacle replacement. The total runtime of the `select_values_from_wall_info ()` stored procedure was found before and after indexing was applied. The mean values were calculated for both scenarios after executing the stored procedure for 5 times. The mean runtime of the `select_values_from_wall_info ()` stored procedure before indexing was 1.770 s whereas the mean runtime of the BIMcache stored procedure after indexing turned out to be 0.287 s.

Cost comparison for index on `select_values_from_wall_info ()` stored procedure

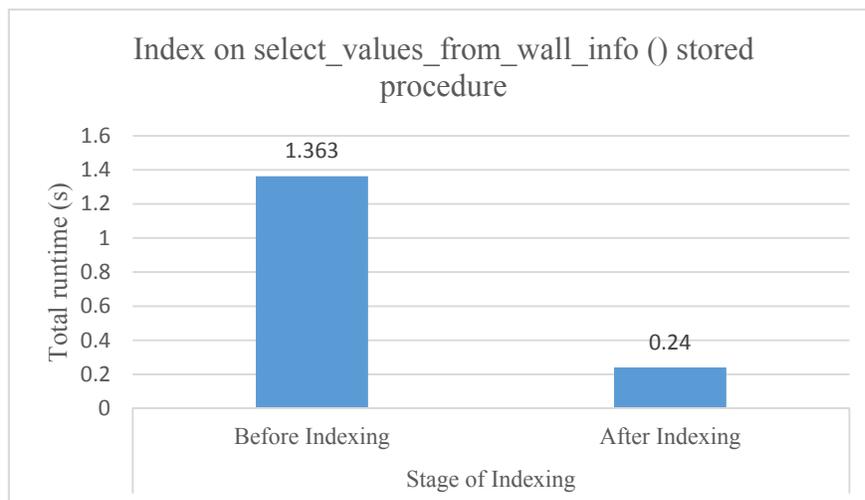


Figure 6.6: Cost comparison for `select_values_from_wall_info` stored procedure

The graph in figure 6.6 illustrates the reduction in the total runtime before and after indexing was applied on the `select_values_from_wall_info` stored procedure.

6.3.1.4 Obtain door information

Stored procedure: `select_values_from_door_info ()`

Gathering information on doors is another important feature in real-time map generation. The door information would be filtered from the AccessBIM model based on the user's X, Y coordinates. The X, Y coordinates of the door's starting and ending points and the current status (whether it is open or close) are the key factors that are required to present door information in a real-time map. Therefore, "select_values_from_door_info()" stored procedure was created and placed with two indexes. Figure 6.7 illustrates the flow of `select_values_from_door_info ()` stored procedure

Flow of the `select_values_from_door_info ()` stored procedure:

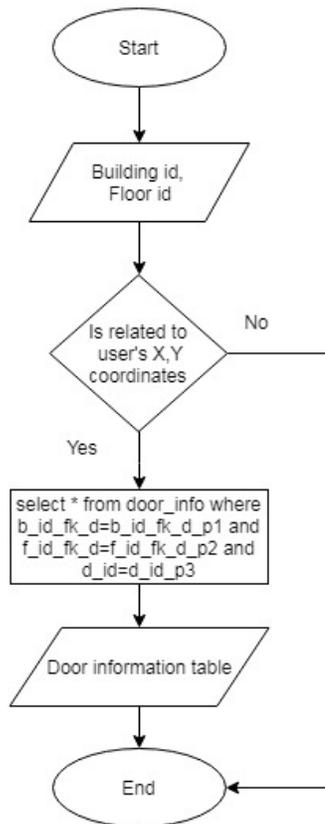


Figure 6.7: Flow of `select_values_from_door_info ()` stored procedure

Index names: `door_info_select_with_building_id`

`door_info_select_with_floor_id`

Implemented indexes:

```
//create index on building id of door_info table to retrieve doors inside specific building
```

```
CREATE INDEX door_info_select_with_building_id ON door_info(b_id_fk_d ASC);
```

```
//create index on floor id of wall_info table to retrieve doors inside specific floor
```

```
CREATE INDEX door_info_select_with_floor_id ON door_info(f_id_fk_d ASC);
```

Results of indexing

The test was conducted with three users after allowing them to navigate to a nearby location, a remote location and undergo an obstacle replacement. The total runtime of the `select_values_from_door_info ()` stored procedure was found before and after indexing was applied. The mean values were calculated for both scenarios after executing the stored procedure for 5 times. The mean runtime of the `select_values_from_door_info ()` stored procedure before indexing was 1.735 s whereas the mean runtime of the `select_values_from_door_info ()` stored procedure after indexing turned out to be 0.360 s.

Cost comparison for index on `select_values_from_door_info ()` stored procedure

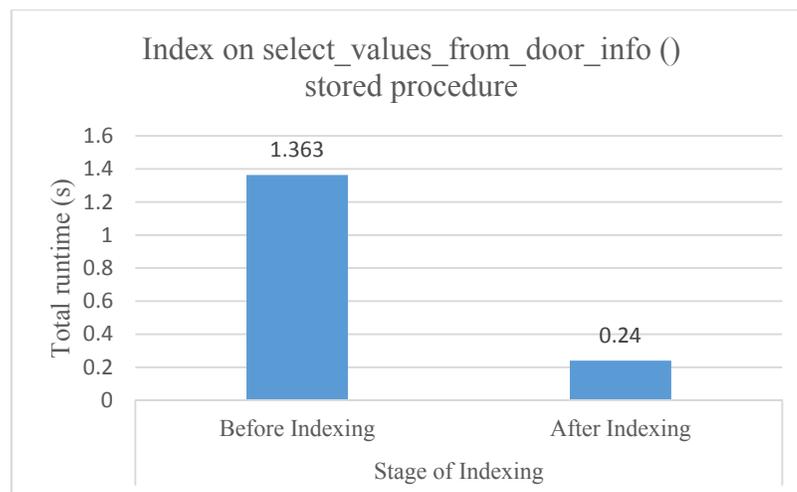


Figure 6.8: Cost comparison for `select_values_from_door_info` stored procedure

The graph in figure 6.8 illustrates the reduction in the total runtime before and after indexing was applied on the `select_values_from_door_info ()` stored procedure.

6.3.1.5 Update door information

Stored procedure: update_door_info ()

Current door information could be retrieved using “select_values_from_door_info” stored procedure. After the user navigates via simulation, if the status of any door changes, it needs to be updated in the database. “update_door_info()” stored procedure was built to increase the efficiency of updating the database. Two partial indexes were placed on the stored procedure to enhance the efficiency. Figure 6.9 illustrates the flow of update_door_info () stored procedure

Flow of the update_door_info () stored procedure:

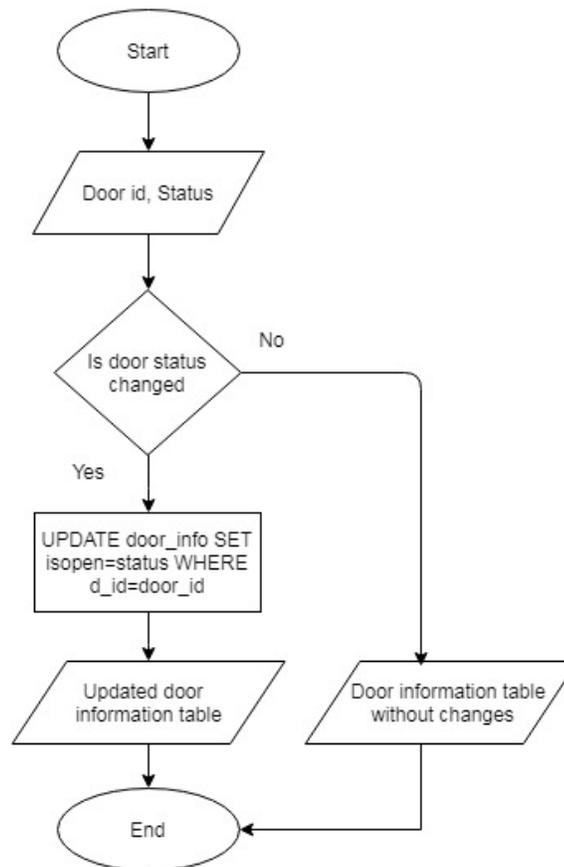


Figure 6.9: Flow of update_door_info () stored procedure

Index names: door_info_get_update_isopen

door_info_get_update_isclosed

Implemented indexes:

```
// create index on door isOpen column to retrieve the doors that are open
```

```
CREATE INDEX door_info_get_update_isopen ON door_info (isopen) WHERE  
isopen =0
```

```
// create index on door isOpen column to retrieve the doors that are close
```

```
CREATE INDEX door_info_get_update_isclosed ON door_info (isopen) WHERE  
isopen=1
```

Results of indexing

The test was conducted with three users after allowing them to navigate to a nearby location, a remote location and with an obstacle replacement. The total runtime of the `update_door_info ()` stored procedure was found before and after indexing was applied. The mean values were calculated for both scenarios after executing the stored procedure for 5 times. The mean runtime of the `update_door_info ()` stored procedure before indexing was 1.745 s whereas the mean runtime of the `update_door_info ()` stored procedure after indexing turned out to be 0.373 s.

Cost comparison for index on `update_door_info ()` stored procedure

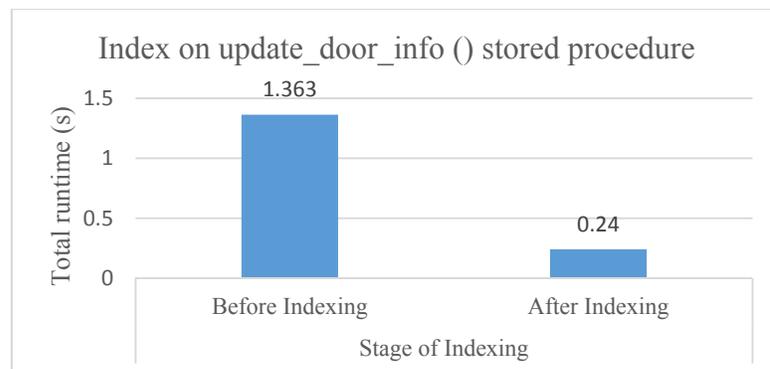


Figure 6.10: Cost comparison for `update_door_info` stored procedure

The graph in figure 6.10 illustrates the reduction in the total runtime before and after indexing was applied on the `update_door_info` stored procedure.

6.3.1.6 Find object information

Stored procedure: `select_object_info()`

Obstacle identification in navigation is a vital factor for real-time map generation. Vision impaired individuals feel comfortable if they can identify obstacles accurately. `select_object_info()` was created to find out obstacles/ objects within a few seconds, that the user meets while navigating. Figure 6.11 illustrates the flow of the `select_object_info()` stored procedure.

Flow of the `select_object_info()` stored procedure:

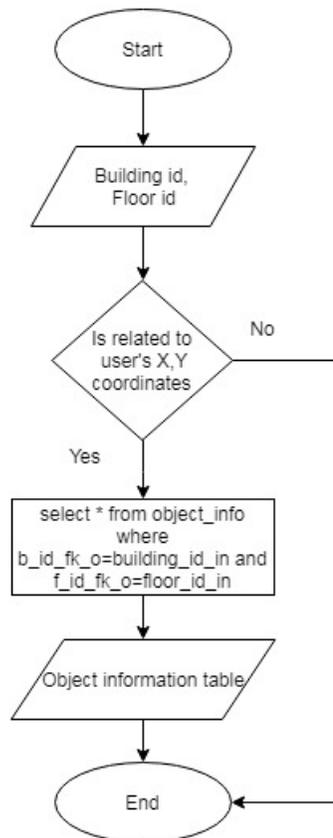


Figure 6.11: Flow of `select_object_info()` stored procedure

Index names: `object_info_select_with_building_id`

`object_info_select_with_floor_id`

Implemented indexes:

```
//create index on building id of object_info table to retrieve objects inside specific building
```

```
CREATE INDEX object_info_select_with_building_id ON object_info(b_id_fk_o  
ASC)
```

```
//create index on floor id of object_info table to retrieve objects inside specific floor
```

```
CREATE INDEX object_info_select_with_floor_id ON object_info(f_id_fk_o ASC)
```

Results of indexing

The test was conducted with three users after allowing them to navigate to a nearby location, a remote location and undergo an obstacle replacement. The total runtime of the `select_object_info ()` stored procedure was found before and after indexing was applied. The mean values were calculated for both scenarios after executing the stored procedure for 5 times. The mean runtime of the `select_object_info ()` stored procedure before indexing was 1.363 s whereas the mean runtime of the `select_object_info ()` stored procedure after indexing turned out to be 0.240 s.

Cost comparison for index on `select_object_info ()` stored procedure

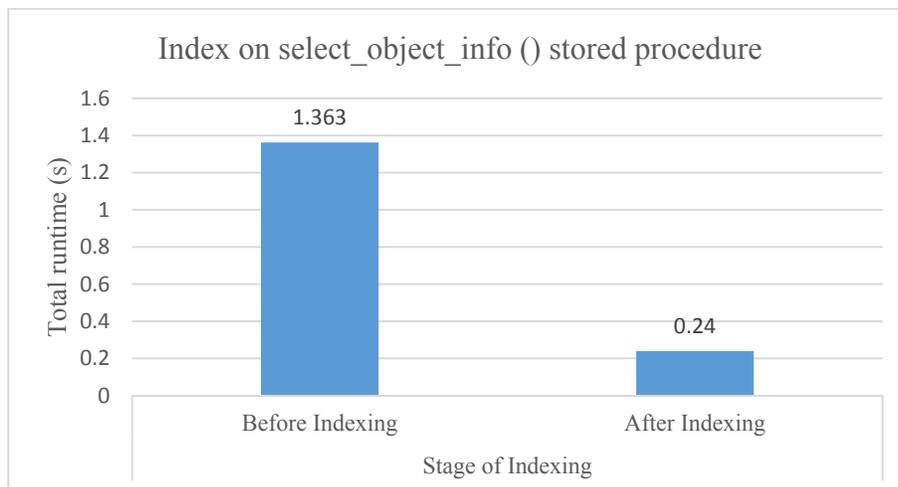


Figure 6.12: Cost comparison for `select_object_info` stored procedure

The graph in figure 6.12 illustrates the reduction in the total runtime before and after indexing was applied on the select_object_info () stored procedure.

6.3.1.7 Update object details

Stored procedure: update_object_info ()

Select object info supports to filter details regarding an object from the database. If the position of any object changes, AccessBIM model should be updated with the most recent information. “update_object_info ()” Stored procedure is there to update modifications according to the user’s most recent navigation. Two indexes were created to enhance the performance of the stored procedure further. Figure 6.13 illustrates the flow of update_object_info () stored procedure

Flow of the stored procedure:

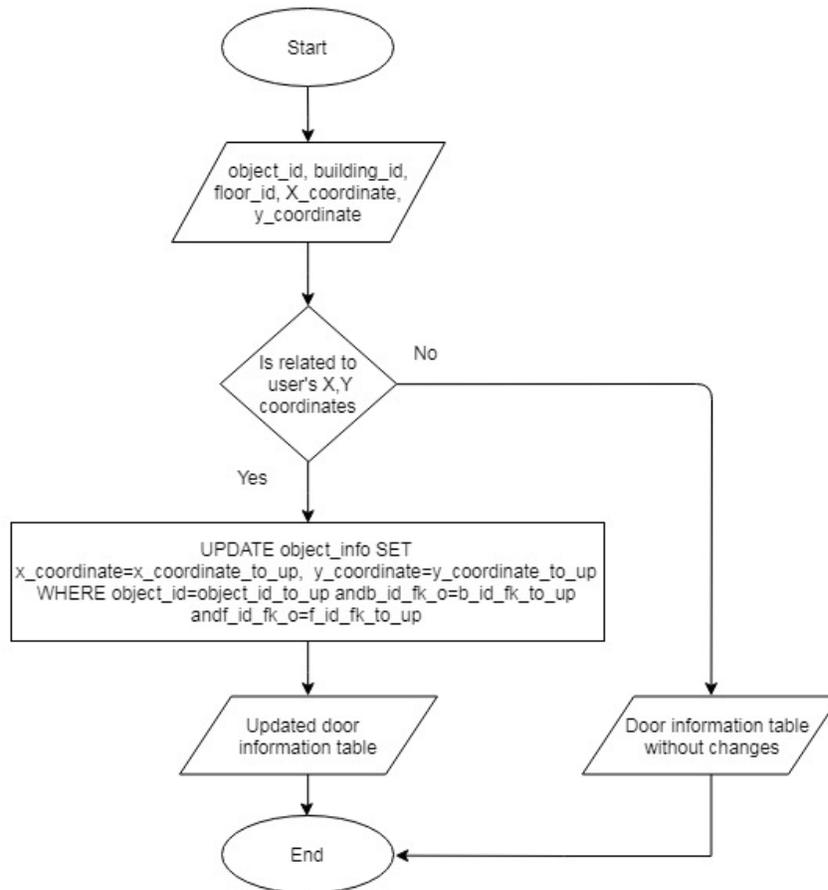


Figure 6.13: Flow of update_object_info () stored procedure

Index names: object_info_update_with_building_id

object_info_update_with_floor_id

Implemented indexes:

```
//create index on building id of object_info table to update objects inside specific building
```

```
CREATE INDEX object_info_update_with_building_id ON object_info(b_id_fk_o  
ASC)
```

```
//create index on building id of object_info table to update objects inside specific building
```

```
CREATE INDEX object_info_update_with_building_id ON object_info(b_id_fk_o  
ASC)
```

Results of indexing

The test was conducted with three users after allowing them to navigate to a nearby location, a remote location and undergo an obstacle replacement. The total runtime of update_object_info () the stored procedure was found before and after indexing was applied. The mean values were calculated for both scenarios after executing the stored procedure for 5 times. The mean runtime of the update_object_info () stored procedure before indexing was 1.557 s whereas the mean runtime of the update_object_info () stored procedure after indexing turned out to be 0.558 s.

Cost comparison for index on update_object_info () stored procedure

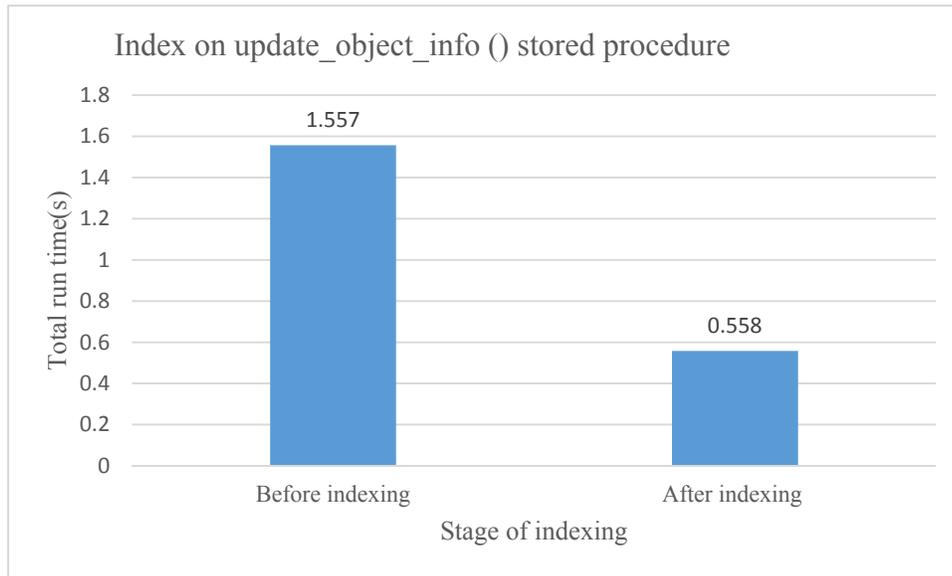


Figure 6.14: Cost comparison for update_object_info stored procedure

The graph in figure 6.14 illustrates the reduction in the total runtime before and after indexing was applied on the update_object_info stored procedure.

6.3.1.8 Find the routes

Stored procedure: get_routing_details_for_a_specific_day ()

Wall, door and object information could be filtered to generate the necessary classes and the map within a lesser time. As the map is generated in real time it is important to identify the user's route to obtain the exact environmental values. Therefore, "get_routing_details_for_a_specific_day ()" stored procedure was created. Consequently, the research project uses a simulator procedure to find route details for a specific day. In an actual environment, it is important to optimise this procedure so that the routes could be found within an hour to generate an efficient real-time map. Reducing the time intervals provides a real-time map with latest environmental characteristics for vision impaired individuals, each time they require a map for indoor navigation. One index on the procedure supports further optimization of data filtering and loading. Figure 6.18 illustrates the flow of the get_routing_details_for_a_specific_day () stored procedure.

Flow of the stored procedure:

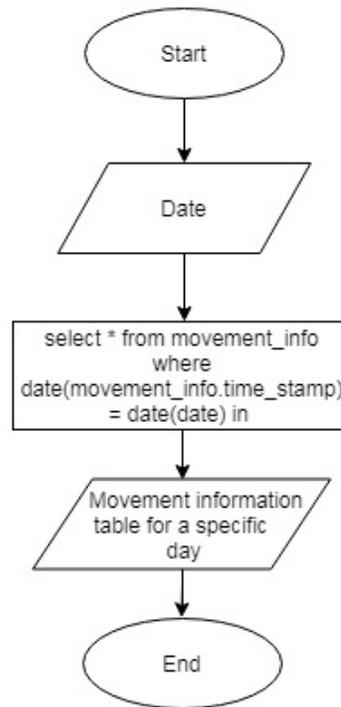


Figure 6.15: Flow of get_routing_details_for_a_specific_day () stored procedure

Index names: movement_info_get_routing_for_date()

Implemented indexes:

```
//create index on date of movement_info table to get specific routings per day
```

```
CREATE INDEX movement_info_get_routing_for_date ON movement_info  
(date(time_stamp));
```

Results of indexing

The test was conducted with three users after allowing them to navigate to a nearby location, a remote location and undergo an obstacle replacement. The total runtime of the get_routing_details_for_a_specific_day () stored procedure was found before and after indexing was applied. The mean values were calculated for both scenarios after executing the stored procedure for 5 times. The mean runtime of the get_routing_details_for_a_specific_day () stored procedure before indexing was 10.776 s whereas the mean runtime of the

get_routing_details_for_a_specific_day () stored procedure after indexing turned out to be 1.337 s.

Cost comparison for index on get_routing_details_for_a_specific_day () stored procedure

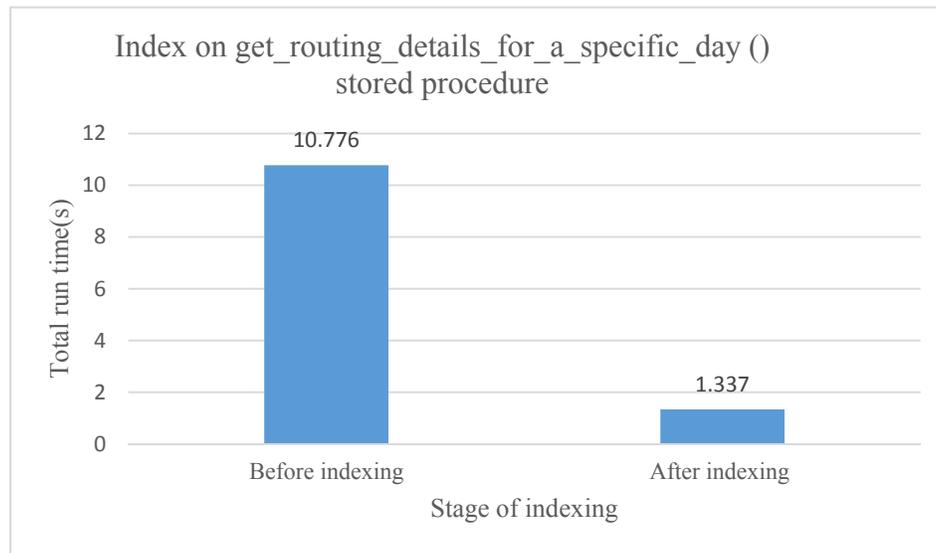


Figure 6.16: Cost comparison for get_routing_details_for_a_specific_day stored procedure

The graph in figure 6.16 illustrates the reduction in the total runtime before and after indexing was applied on the get_routing_details_for_a_specific_day stored procedure.

6.3.2 Performance Enhancement with the use of BIMcache for Real-Time Indoor Map Generation

BIMcache is a database optimization technique used in this research to enhance the performance of the AccessBIM database. This section demonstrates how BIMcache facilitates the generation of the indoor map in real-time with its test results described in section 6.4.2.

a) Scenario 1

Name: The user navigates to a nearby destination (Main entrance to Lab 702-Test Simulation). The user needs to navigate from the main entrance of the 7th floor (A) to the Lab 702 (B) which is a nearby location.

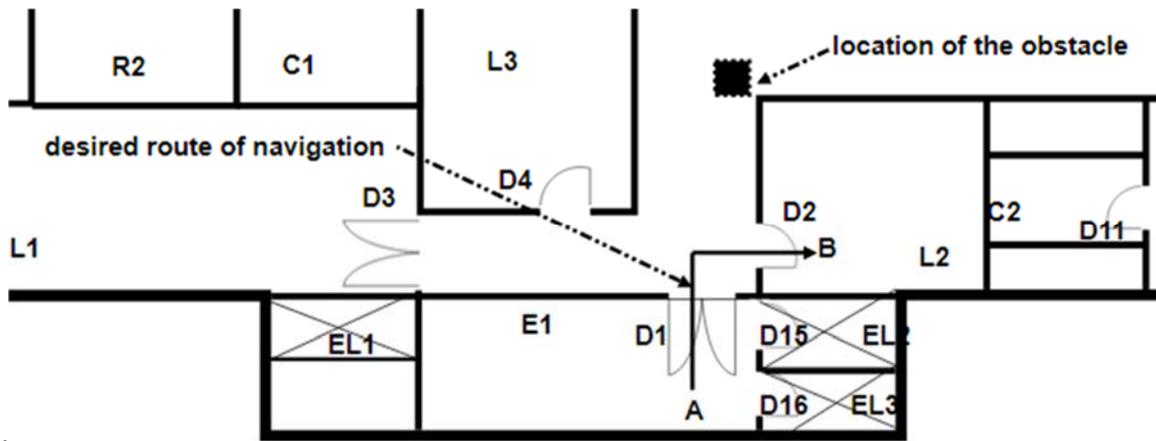


Figure 6.17: Floor arrangement for first-time navigation

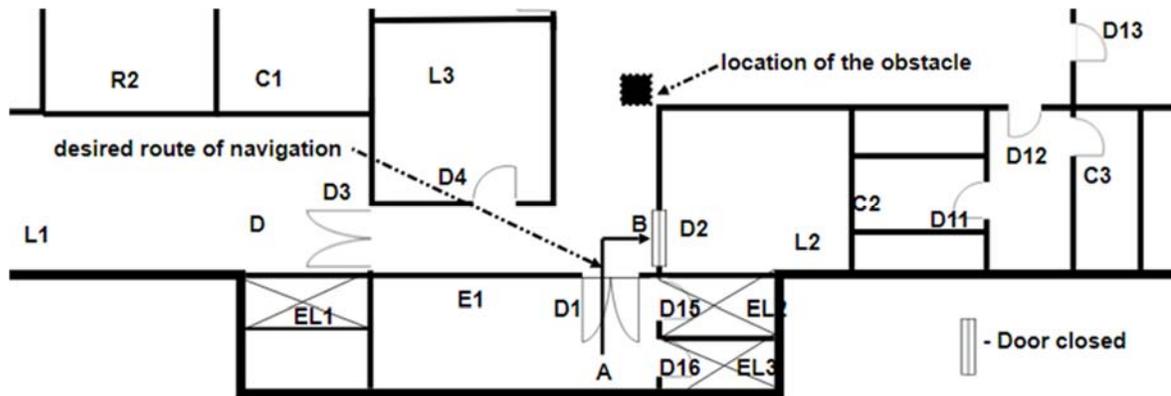


Figure 6.18: Floor arrangement for second-time navigation.

Expected output:

The author expects that, when a vision impaired individual selects the source and destination:

- The simulator would check BIMcache.
- If the selected route is unavailable in BIMcache, the user is allowed to navigate within the simulator and the movement information of the user and the selected source and destination is stored in the database.
- When a new user selects the same source and destination, a map will be loaded via the existing movement information available in the BIMcache.

- The user can refer the map to navigate within the indoor environment and if there is any change such as the movement of an obstacle or change in the status of a door, it will be updated in the BIMcache movement information for future use.

Test results:

Test results for first-time navigation:

-Simulator checks the availability of a route for the given source and destination within the BIMcache data as shown in figure 6.19.

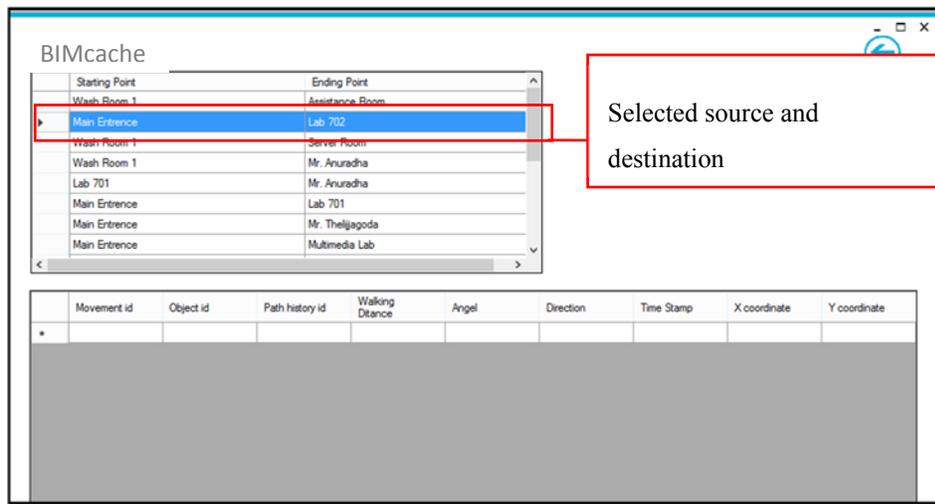


Figure 6.19: Check route availability in BIMcache data

-Allows the user to navigate within the simulator due to the unavailability of movement information in the BIMcache data.

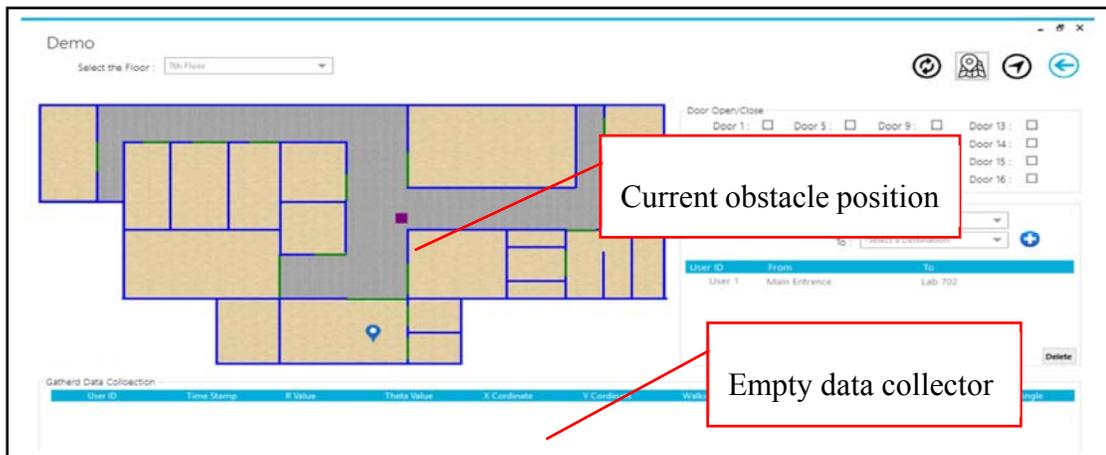


Figure 6.20: User navigate to the desired destination within simulator

-Generate real-time map after the navigation

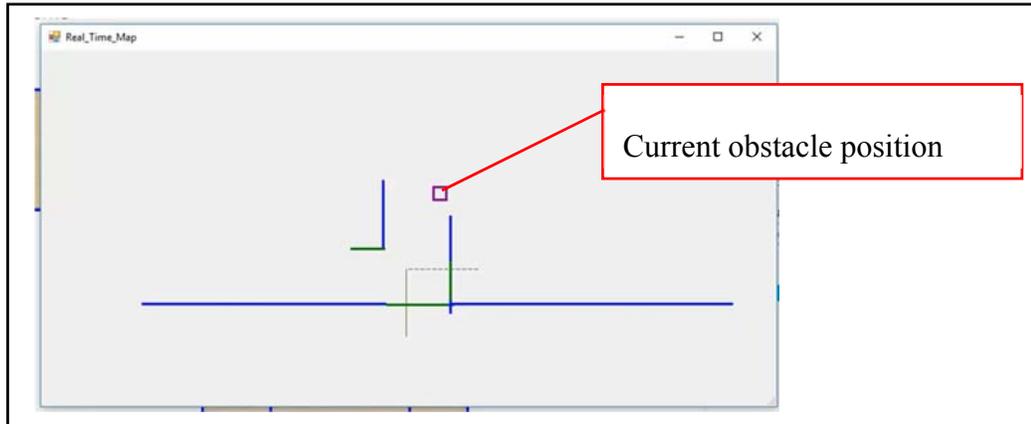


Figure 6.21: Generated real-time map

-Add movement information of scenario 1 to the BIMcache.

-Load movement information when user 2 selects the same source and destination for navigation.

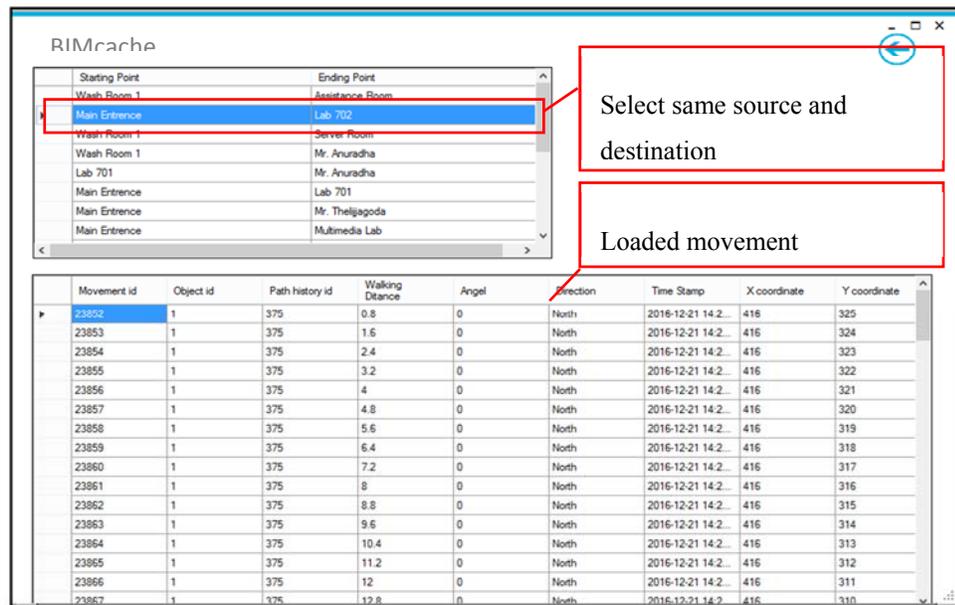


Figure 6.22: Loaded movement information for scenario 1

- Generate real-time map based on the loaded movement information

- Allow the user to navigate with the help of the generated map.

- Update the BIMcache movement information with the object replacement and door status.

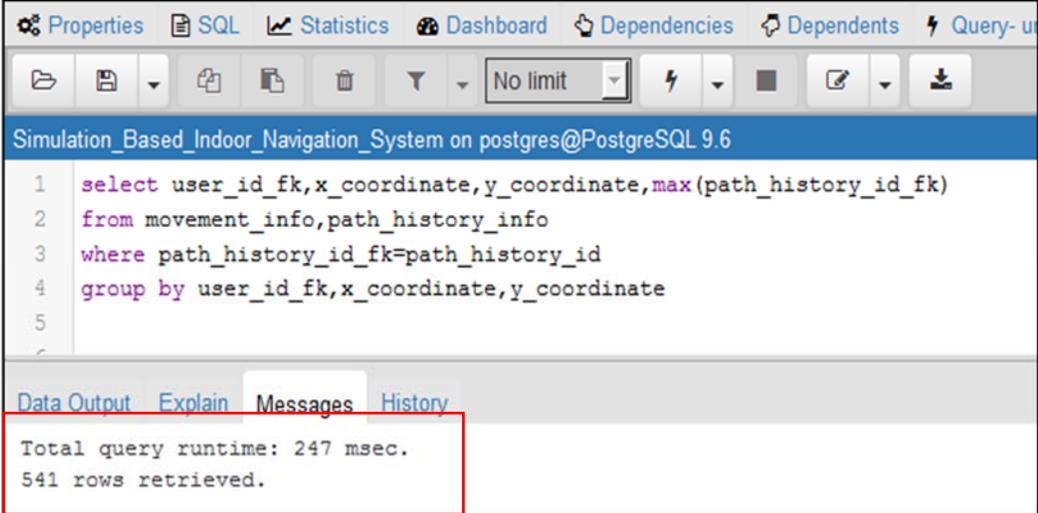
6.3.3 Performance Enhancement with the use of Query Rewriting for Real-Time Indoor Map Generation

Query rewriting was used throughout the real-time map generation project to obtain the most relevant data from the collected data set through the use of The SQL JOIN clause. This section demonstrates how query rewriting facilitates the generation of the indoor map in real-time with its test results described in section 6.4.3.

i.e.: To obtain the real-time map, data from both movement_info and path_history_info tables, are required.

a) Scenario 1

Figure 6.23 demonstrates a situation where query rewriting is not used. Highlighted area of the figure proves that without the SQL JOIN condition, it is unable to filter the most recent path history information to generate the real-time map.



The screenshot shows a PostgreSQL query execution window. The query is as follows:

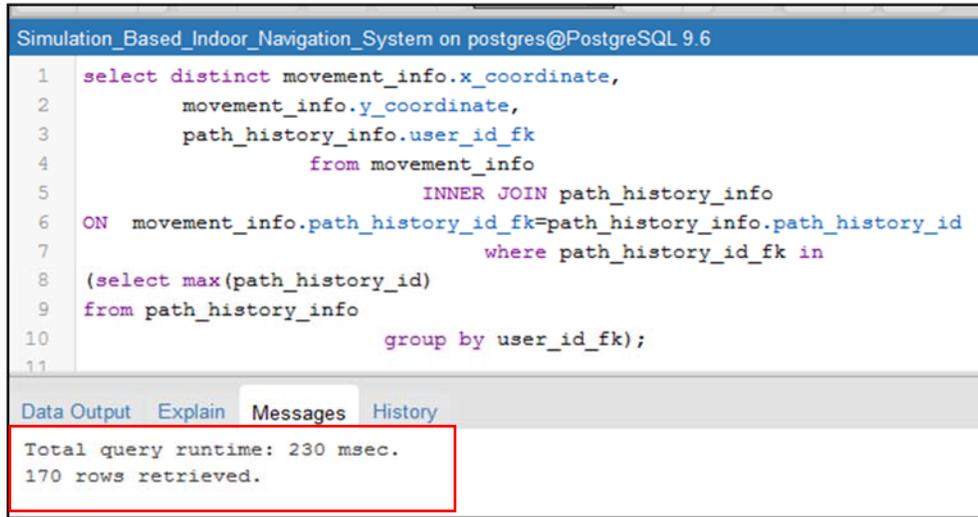
```
1 select user_id_fk,x_coordinate,y_coordinate,max(path_history_id_fk)
2 from movement_info,path_history_info
3 where path_history_id_fk=path_history_id
4 group by user_id_fk,x_coordinate,y_coordinate
5
```

The output of the query is displayed in the 'Data Output' tab, which is highlighted with a red box. The output shows:

```
Total query runtime: 247 msec.
541 rows retrieved.
```

Figure 6.23: Output of query execution without SQL JOIN

b) Scenario 2



```
Simulation_Based_Indoor_Navigation_System on postgres@PostgreSQL 9.6
1  select distinct movement_info.x_coordinate,
2     movement_info.y_coordinate,
3     path_history_info.user_id_fk
4     from movement_info
5         INNER JOIN path_history_info
6 ON  movement_info.path_history_id_fk=path_history_info.path_history_id
7     where path_history_id_fk in
8     (select max(path_history_id)
9     from path_history_info
10    group by user_id_fk);
11
```

Data Output Explain Messages History

Total query runtime: 230 msec.
170 rows retrieved.

Figure 6.24: Output of query execution with SQL JOIN

Figure 6.24 is an example of a situation, which uses query rewriting. Highlighted area of the figure proves that with the use of the SQL JOIN condition it is possible to filter the most relevant details for map generation within a short period of time.

6.4 Comparison of Results and Discussion

The research on Vision Impaired Indoor Navigation and Real-time map generation utilized several database optimization techniques such as stored procedures, Indexing, BIMcache and Query rewriting. The previous section described how each of these database optimization techniques were utilized in the research. This section discusses the results of database optimization apart from comparing the total time taken for real-time map generation. It also describes the quantitative benchmark for performance enhancement.

6.4.1 Discussion on stored procedures and indexing

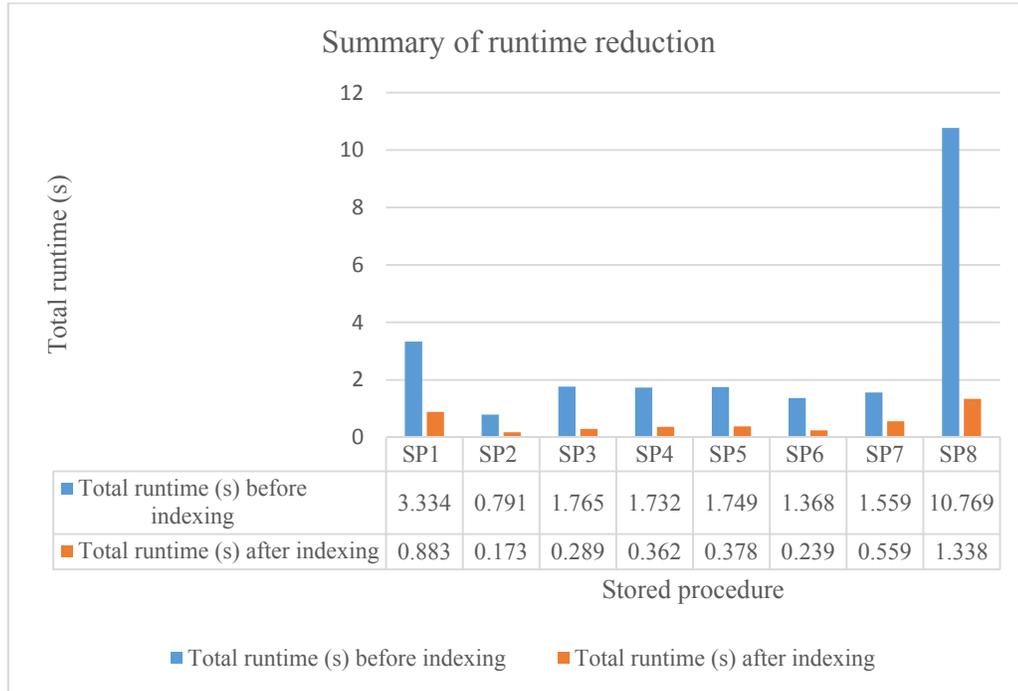


Figure 6.25: Summary of time reduction comparisons

The use of indexes facilitate in reducing the execution time of the stored procedures in inserting, updating, deleting and retrieving data. The same map generation instance was executed for 8 times by applying indexing on one stored procedure at a time for which the results are presented in figure 6.25. It can be observed from figure 6.25 that the use of indexing has reduced the execution time of each instance by a substantial amount.

As per the results indicated in figure 6.25 the addition of the total runtime of executing each stored procedure without indexing adds up to 23.077 s while the sum of the total runtime of executing each stored procedure with indexing totals to 4.22 s. However, as the author has considered each stored procedure individually, the total run time values represent the sum of the individual changes of applying indexing to each stored procedure. Besides, indexing that supports one operation may slow down another. Thus, to overcome this problem a single map generation instance was executed by applying indexing on all the 8 stored procedures at once and their execution times were noted. The graph given below depicts the total runtime for executing all the 8 stored procedures at once, before and after indexing was applied.

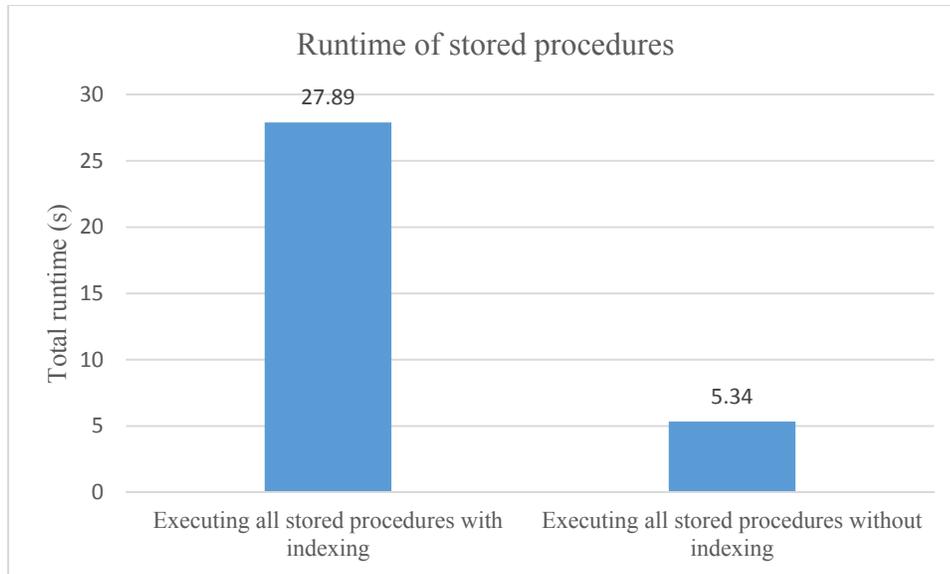


Figure 6.26: Execution time of all 8 stored procedures with and without indexing

When indexing was applied on all the 8 stored procedures at once, the total run time amounted to 5.34 s while the total run time of all 8 stored procedures before indexing was applied accounted to 27.89 s. Hence the percentage of database optimization due to stored procedures and indexing can be calculated as a percentage of an index efficiency factor (E_f).

The following index efficiency calculation is based on the output illustrated in figure 6.46

$$T_n = \sum_{i=1}^n a_i \quad \text{Equation 1}$$

n = Number of stored procedures

a = Summation element

n_w = Number of stored procedure with indexes

i = Index counter

T_n = Total execution time without indexing

T_{nw} = Total execution time with indexing

E_f = Index efficiency factor

According to figure 6.42:

$$T_n = 27.89 \text{ ms}$$

$$T_{nw} = \sum_{i=1}^{n_w} a_i \quad \text{Equation 2}$$

$$T_{nw} = 5.34 \text{ ms}$$

$$E_f = T_n / T_{nw}$$

Equation 3

$$E_f = \frac{27.89 \text{ ms}}{5.34 \text{ ms}} = 5.2228 \approx 5.2$$

6.4.2 Discussion on the use of BIMcache

The use of BIMcache reduces the time of query execution and increases performance of the database as the real-time map will be automatically loaded to the user if any previous navigation to the same destination from the same source has taken place. Table 6.1 illustrates the test results of the queries executed with and without the use of BIMcache.

Table 6.1: Query execution time with and without the use of BIMcache

Number_of_users (n)	Execution_time without the use of BIMcache (s)	Execution_time with the use of BIMcache (s)
1	15.926	8.613
2	17.605	9.826
3	19.459	8.303
4	17.918	8.835
5	16.994	9.335
6	11.506	9.824
7	15.314	8.999
8	17.138	7.844
9	14.866	8.47
10	14.619	9.367
11	11.213	9.976
12	17.625	8.589
13	14.724	9.978
14	16.253	9.598
15	13.354	7.943

It can be observed in figure 6.27 that, the use of BIMcache has significantly reduced the time of query execution.

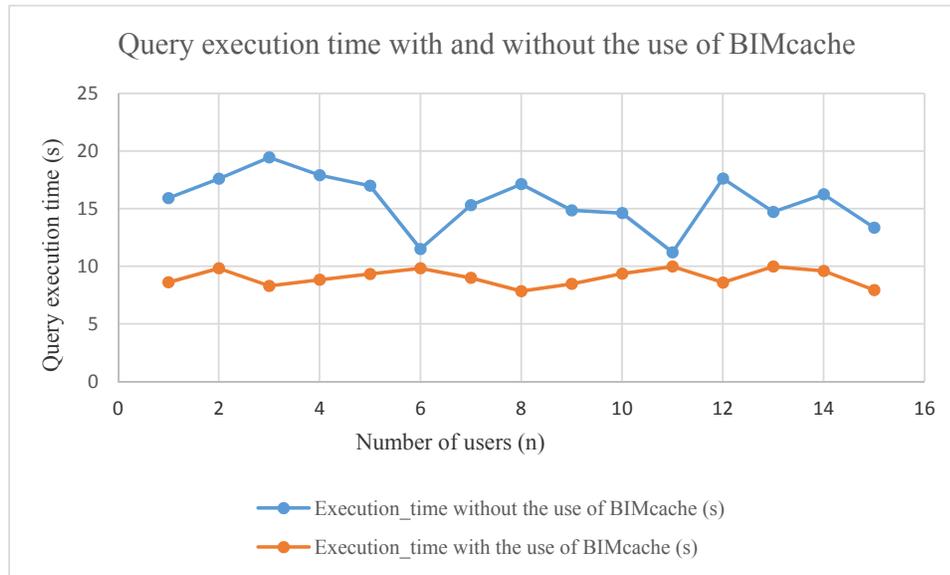


Figure 6.27: Query execution time with and without the use of BIMcache

6.4.3 Discussion on query rewriting

As illustrated in section 6.3.3 a query to find the most recent path history information was executed with and without a SQL JOIN to assess the efficiency of query rewriting. The query executed without the SQL JOIN retrieved 541 rows of most recent path history information within 247 ms while the query with the SQL JOIN retrieved only 170 rows of most recent path history information within 230 ms. This indicates that the use of a SQL JOIN increases the efficiency of the database by minimizing the execution time by reducing the number of rows retrieved. Hence, the use of a SQL JOIN facilitates real-time map generation by retrieving the most relevant information from the database within a short period of time.

The number of data rows retrieved within a specific period determines the Optimization factor (O_f). The Optimization factor analyzes the proportion of efficiency of query rewriting in terms of a numeric value. The Optimization factor of the query to find the most recent path history information executed with and without the SQL JOIN is given below.

$$T_q \propto N_d$$

$$T_q = O_f \times N_d \quad \text{Equation 4}$$

$$O_f = \frac{T_q}{N_d} \quad \text{Equation 5}$$

The following calculations were performed using equation 7;

- Optimization factor for the query executed without the SQL JOIN (O_{f1})

$$O_{f1} = \frac{247 \text{ ms}}{541 \text{ rows}} = 0.4565 \approx 0.46$$

- Optimization factor for the query executed with the SQL JOIN (O_{f2})

$$O_{f2} = \frac{230 \text{ ms}}{170 \text{ rows}} = 1.3529 \approx 1.36$$

T_q = Query execution time

N_d = Number of retrieved data rows

O_f = Optimization factor

O_{f1} = Optimization factor for scenario 1

O_{f2} = Optimization factor for scenario 2

Equation 5 shows the relationship between the execution time and the number of rows corresponding to the query execution with and without a SQL JOIN. The Optimization factor for the query executed without the SQL JOIN (O_{f1}) is 0.46 whereas the Optimization factor for the query executed with the SQL JOIN (O_{f2}) is 1.36. Since the Optimization factor for the query with the SQL JOIN is greater than the Optimization factor for the query without the SQL JOIN ($O_{f2} > O_{f1}$), it can be justified that the use of SQL JOINS enhances the efficiency of the AccessBIM database.

6.4.4 Comparison of the total time in real-time map generation

The key factor of monitoring the system performance indicates that the implemented AccessBIM database optimization model is successful in building a real-time map based on proposed algorithms in a simulated environment.

The database is responsible for executing queries; hence, priority must be given to efficient query execution as a factor of time. Since the real-time map is generated based on the inputs of environmental changes, the comparison of the total time in real-time map generation was calculated for 15 occurrences of user navigations. The total time taken for map generation (T_{mp_tot}) depends on the following time factors used in equation 8.

1. Time which takes a user to navigate within the simulator (T_{u_nav})
2. Time taken to process the collected data ($T_{process}$)
3. Time to generate the real-time map based on collected data (T_{mp_gen})

$$T_{mp_tot} = T_{u_nav} + T_{process} + T_{mp_gen} \quad \text{Equation 6}$$

The total map generation time can be calculated by changing the following parameters to evaluate the strength of the AccessBIM framework.

1. Indoor map generation time without database optimization
2. Map generation time with query rewriting
3. Map generation time with stored procedures and indexing
4. Map generation time with BIMcache
5. Map generation time with all 3 optimization techniques: query rewriting, stored procedures & indexing and BIMcache

The analysis given below was conducted to further signify the efficiency of the AccessBIM database in a simulated environment.

In order to analyze the efficiency of stored procedures and query rewriting for performance enhancement in real-time map generation, fifteen scenarios were selected for the analysis with increasing number of users. Each of the 15 scenarios were run for 5 times and the mean time for usual execution, with query rewriting, with stored procedures and indexing together with BIMcache were calculated. All the times were measured in seconds.

Table 6.2: Analysis based on Total real-time map generation time

Number of Users	Original execution time (Ot)	With query rewriting (Or)	With query rewriting + stored procedures & indexing (Oi)	With query rewriting + stored procedures & indexing + BIMcache (Om)
1	72.18	17.86	13.20	34.24
2	180.46	44.65	33.00	46.13
3	400.29	99.03	73.20	42.15
4	442.94	109.58	81.00	27.93
5	482.31	119.33	88.20	31.36
6	656.21	162.35	120.00	20.64
7	725.11	179.39	132.60	33.77
8	787.45	194.82	144.00	38.55
9	790.73	195.63	144.60	47.17
10	803.85	198.88	147.00	35.27
11	816.98	202.12	149.40	46.59
12	823.54	203.75	150.60	29.82
13	987.59	244.33	180.60	36.49
14	1023.68	253.26	187.20	26.92
15	1033.53	255.70	189.00	44.56

Table 6.2 indicates that the total time taken to generate the real-time map using all three forms of database optimization is greater than the total time taken to generate the map using only stored procedures + indexing and query rewriting, in the first two instances. This is mainly due to the fact that when few users request navigation assistance less crowdsourced data is available in the database thereby taking a greater time to generate the real-time map. However, when more and more users navigate, more and more crowdsourced information on specific locations can be obtained. Hence, the real-time map takes less time to be generated as the

queries are processed much faster as data is already available in the AccessBIM database. Thus, it can be concluded that the total map generation time reduces with the increasing number of users.

The graph shown in figure 6.28 was plotted in accordance to the total real-time map generation times to show the effectiveness of using stored procedures, BIMcache and query rewriting in real-time map generation.

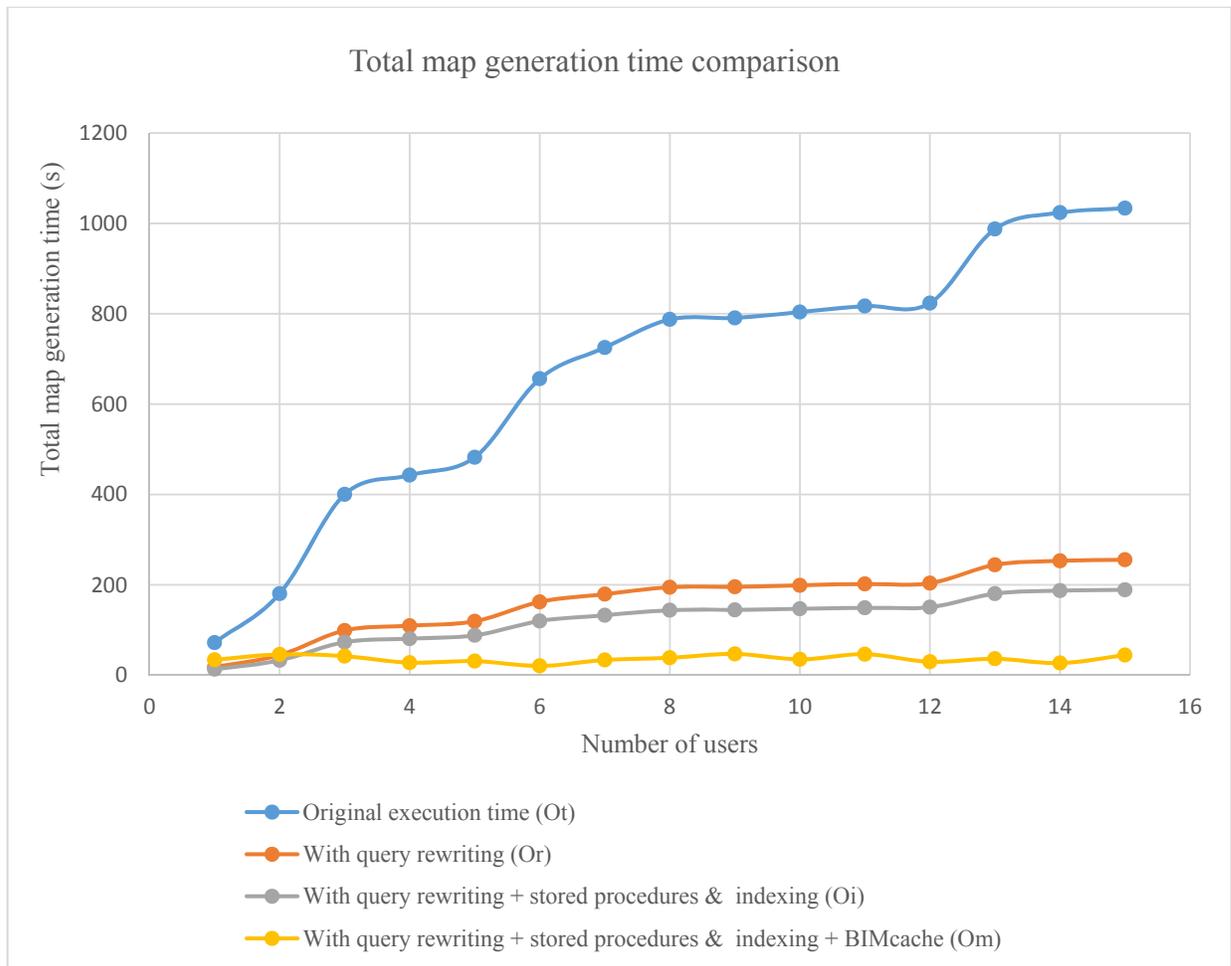


Figure 6.28: Total real-time map generation time comparison

Figure 6.28 compares the usual time taken to generate the real-time map with the total time taken to generate the real-time map with the use of query rewriting, BIMcache, stored procedures and indexing. The total map generation time has reduced by a considerable amount with the application of stored procedures and query rewriting. It is evident that the use of

BIMcache has significantly reduced the total time taken to generate the real-time map by a substantial amount. The use of algorithms discussed in chapter four makes a significant impact on the total real-time map generation time by reducing the time of query execution.

Table 6.3 given below summarizes the results of all the tests performed on the AccessBIM framework. The research results indicate that the synthesis of the three database optimization techniques has significantly reduced the total map generation time facilitating the indoor map to be generated in real-time.

Table 6.3: Summary of research results

Element	Details	Results	
		Without optimization	With optimization
Stored procedures & indexing	A map generation instance was executed for 8 times by applying indexing on one stored procedure at a time.	23.077 s	4.22 s
	A single map generation instance was executed by applying indexing on all the 8 stored procedures at once.	27.89 s	5.34 s
BIMcache	The query execution time when a single map generation instance was executed with 5 users requesting for navigation assistance simultaneously.	16.994 s	9.335 s

Query rewriting	The query execution time when the queries are executed with and without a SQL join.	Retrieved 541 rows within 247 ms	Retrieved 170 rows within 230 ms
Query execution time	25000 queries from the movement_info table related to map generation was executed with and without the use of query rewriting, BIMcache, stored procedures and indexing with 5 users requesting for navigation assistance simultaneously.	16.994 s	10.016 s
Total map generation time	The total map generation time when a single map generation instance was executed with 5 users requesting for navigation assistance	482.31 s	313.64 s

6.4.5 Quantitative benchmark for performance enhancement

It is crucial that vision-impaired individuals navigate safely through indoor environments without collisions. Hence, faster retrieval of data is essential to generate a reliable indoor map. This section mainly focuses on establishing a benchmark to assess the performance of the AccessBIM database that facilitates faster data transactions. Linear association was used to evaluate the performance of the AccessBIM database where two numerical variables were used to demonstrate and forecast the dependent variable [77].

In order to test the performance of the AccessBIM model, 25000 records were used from the movement_info table demonstrated in chapter 3. The queries were tested and simulated using the PostgreSQL Graphical User Interface (GUI) using an Intel® Core™ i5 processor with an

8GB memory. Real-time Indoor Navigation simulation engine was used to test the performance of the AccessBIM framework. The test results have been obtained by the simultaneous execution of queries in the database by changing the number of simultaneous executions from 1 to 50 queries at a time. The PostgreSQL command EXPLAIN ANALYZE [68] was used to measure the query execution time (T_{q_exe}) of the above mentioned simultaneous executions.

PostgreSQL formulates a query plan for each query it receives. The EXPLAIN command is used to monitor the query plan for each query executed. The EXPLAIN ANALYZE command shows additional execution statistics such as the plan node execution times and rows counts [78].

According to the gathered test results, Execution time (T_{q_exe}) is taken to predict the performance enhancement, which will act as the dependent or the response variable of the linear association. Number of Occurrences resembles the number of users (n), which is the predictor or else the independent variable that was used to forecast the performance.

The table in Appendix D indicates the test results collected by executing 25000 queries related to map generation among 100 users. (100 occurrences) Table 6.4 demonstrates the execution time for the first 15 occurrences without the use of query rewriting, BIMcache, stored procedures and indexing. Table 6.5 demonstrates the execution time for the first 15 occurrences with database optimization techniques such as query rewriting, BIMcache, stored procedures and indexing. Both the tables contain two columns named “Number of users” (n) and “Execution time” (T_{q_exe}) which represents the X-axis and the Y-axis of the graph drawn to represent the linear relationship between the dependent and the independent variable. In order to demonstrate a clear relationship, the graphs have been drawn to represent the execution times of all the 100 occurrences (Appendix D) even though the tables given below illustrate only the first 15 occurrences.

Table 6.4: Comparison between the number of occurrences and execution time without optimization

Number_of_users (n)	Execution_time (ms) <i>(T_{q_exe})</i>
1	15.926
2	17.605
3	19.459
4	17.918
5	16.994
6	11.506
7	15.314
8	17.138
9	14.866
10	14.619
11	11.213
12	17.625
13	14.724
14	16.253
15	13.354

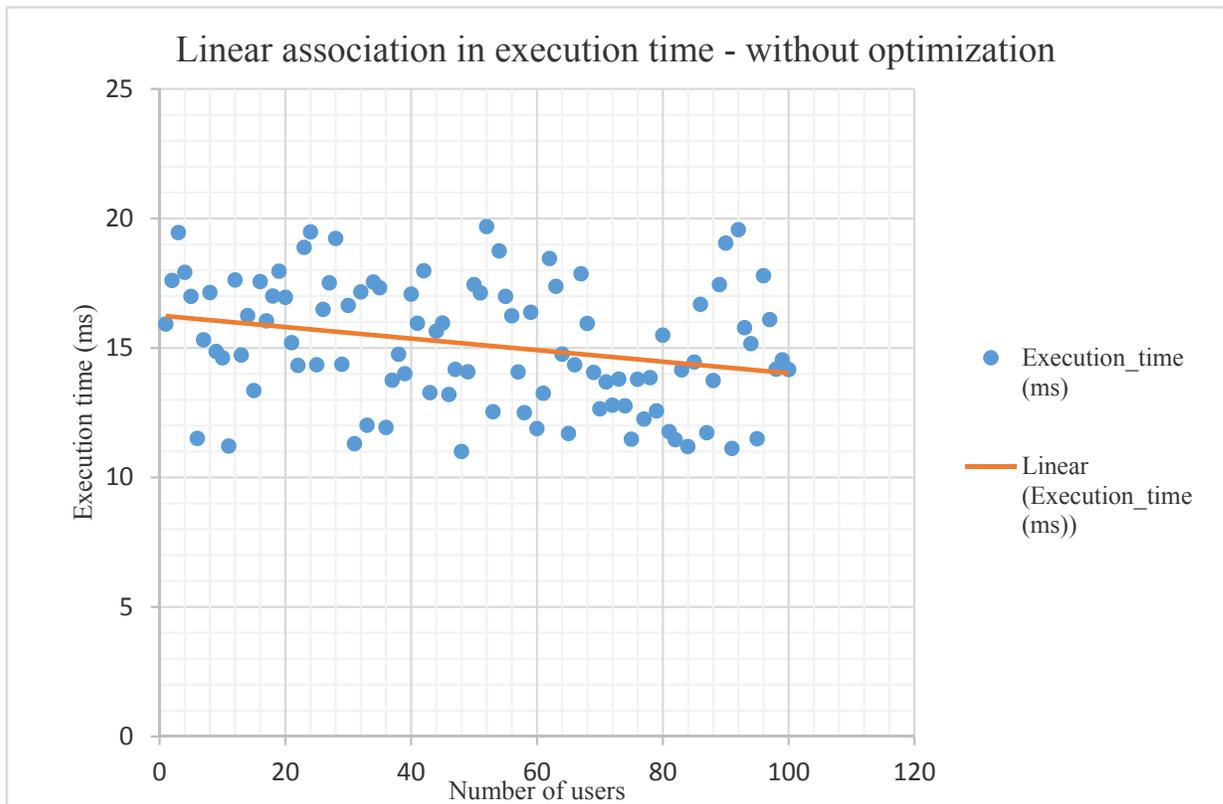


Figure 6.29: Linear association between number of users and execution times (without optimization)

Figure 6.29 indicates the linear association between the number of users and the execution times without optimization. The regression equation obtained from the linear association generates the below-mentioned quantitative benchmark:

$$T_{q_exe} = 16.256 - 0.0233 n \quad \text{Equation 7}$$

Where;

T_{q_exe} : Execution time in ms

n : Number of users

The equation given above inherits the pattern of the linear association as $Y = \beta_0 + \beta_1 X$. β_0 exhibits the intercept of the line whereas β_1 indicates the slope of the line. According to equation 9, $\beta_0 = 16.256$ and $\beta_1 = -0.0233$. The equation can be used to predict the execution time when the number of users are known.

In order to compare the significance of the AccessBIM framework, the same queries were executed along with several optimized strategies such as query rewriting, BIMcache, stored procedures and indexing. Table 6.5 displays the execution times of the first 15 occurrences with optimization.

Table 6.5: Comparison between number of occurrences and the execution time with optimization

Number_of_users(n)	Execution_time(ms) <i>(T_{q_exe})</i>
1	8.0097
2	10.986
3	11.28
4	8.3745
5	10.016
6	9.89
7	12.242
8	11.841
9	10.005
10	10.816
11	8.507
12	14.392
13	11.677
14	12.365
15	11.04

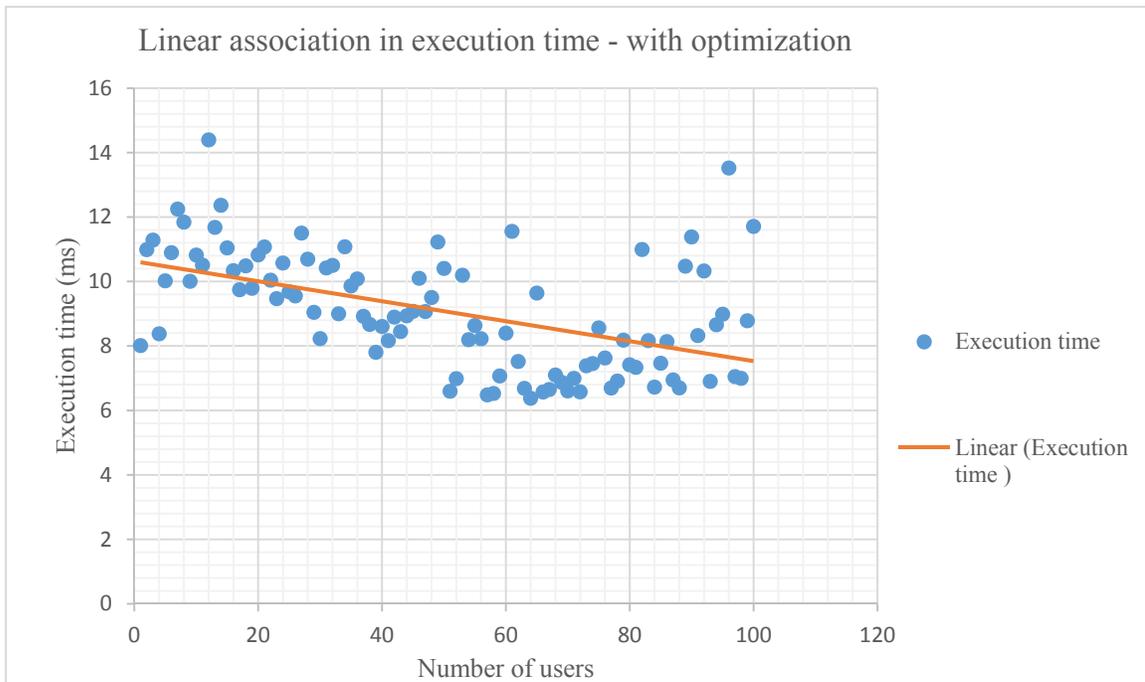


Figure 6.30: Linear association between number of users and execution time (with optimization)

Figure 6.30 indicates the linear association between the number of users and the execution times with optimization. The regression equation obtained from the linear association generates the below-mentioned quantitative benchmark:

$$T_{q_exe} = 10.62 - 0.03092 n \quad \text{Equation 8}$$

Where;

T_{q_exe} : Execution time in ms

n : Number of users

Here too the dependent variable Y (T_{q_exe}), is related to the independent variable X (n). β_0 exhibits intercept of the line whereas β_1 indicates the slope of the line. According to equation 11, $\beta_0 = 10.62$ and $\beta_1 = -0.03092$.

The slope of both the graphs (figure 6.29 & figure 6.30) being a negative value indicate that the time of query execution reduces with the number of users. This is mainly due to the fact that when few users request navigation assistance less crowdsourced data is available in the database thereby taking a greater time to generate the real-time map. However, when more

users navigate, more crowdsourced information on specific locations can be obtained thereby enriching the data in the AccessBIM database with relevant and new information. The figures 6.29 and 6.30 indicate the linear association between the number of users and the query execution times with the use of database optimization. The slope of the graph being negative indicate that the time of query execution reduces with the increasing number of users as the queries are processed much faster given that data is already available in the AccessBIM database.

In addition, the author would like to express that the simulated users are performing the task of navigation from a set of starting points to a set of destinations, one set per each occurrence. This set is a random selection on the simulation environment and the path variables of each navigation set is collected into the database and updated when changes occur. A change occurs in an event of an obstacle in the path such as an object, closed door, etc. As it is illustrated in the above figure the user data vector is a vital factor in the database optimization process. The performance enhancement benchmark generated to predict the execution time in this research can be used to assess the performance of any other relational databases as we

6.5 Evaluation of the AccessBIM framework against existing work

6.5.1 Comparison between query execution time and size of data retrieved

Mithani et al. paper on ‘A Novel Approach for SQL Query Optimization’ [79] compares the execution time (sec) and the size of the data retrieved (MB) for both optimized and non-optimized queries. Figure 6.31 illustrates the output of the experiment performed in executing the actual query and the optimized query by varying the size of data retrieved. The results of the experiment depict that both the optimized queries and the non-optimized queries has had a similar execution time.

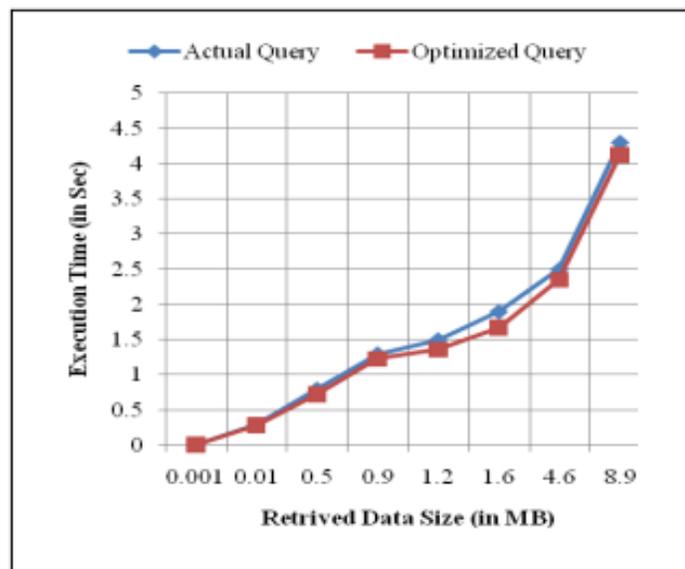


Figure 6.31: Query execution with respect to time and size
(Taken from: ‘A Novel Approach for SQL Query Optimization’ [79])

Mithani et al. followed an approach where an input query was run first to acquire information on the database schema, number of joins, missing indexes and the redundant use of tables from which syntax, constraint information and join aggregation can be identified. This output was then used to rewrite the input query in an optimized mode. The new input query is used to generate a detailed query execution plan to identify any existing performance pitfalls. The optimized execution plan would include SQL tuning, hash temporary table, join benchmark and summarized data.

Due to the absence of technically identical research work for evaluation, the results of Mithani et al. paper was used as a benchmark to assess the degree of performance of the AccessBIM framework as both models contained quite a few technical similarities despite its optimization approach. The similarities between the two frameworks are described in table 6.6.

Table 6.6: Technical comparison of the two models

	Variable	Framework described in the referenced paper [78]	AccessBIM framework
Similarities	Processor	Core i5-2430M	
	System type	64 bit operating system	
	Clock frequency	2.4 GHz	
	RAM	4 GB	
Differences	Operating system	Windows 7	Windows 8.1
	DBMS	MySQL	PostgreSQL

The work in Mithani’s paper was done in a windows 7-64 bit machine that uses a Core i5-2430M CPU with a clock frequency of 2.4GHz and 4GB of RAM while the work in this thesis was performed in a windows 8, 64 bit machine that uses the same Intel core i5-2430M processor with a 4 GB RAM. It is presumed that the difference in the version of the operating system and DBMS has no striking impact on query execution as both MySQL and PostgreSQL are world renowned, time proven enterprise open source relational database management solutions. Hence, the performance of the AccessBIM framework could be determined by comparing the execution time of the queries executed through the AccessBIM framework with the execution times of this research.

For the comparison, three optimized queries were executed based on the movement_info table with varying data sizes and their execution times were noted. These values were then compared to three specific points on the graph denoted in figure 6.32.

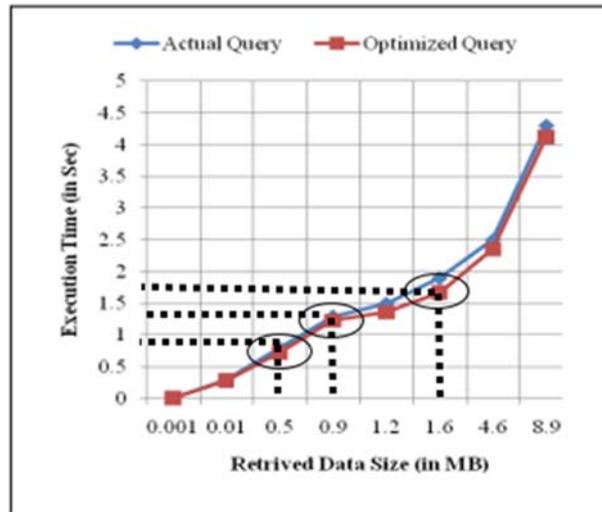


Figure 6.32: Query execution with respect to time and size

```

select distinct path_history_id_fk

from movement_info

where path_history_id_fk in (select distinct path_history_id_fk from
movement_info where time_stamp > CURRENT_TIMESTAMP - interval '15
minutes')

order by path_history_id_fk asc;

```

Figure 6.33: Sample query

Figure 6.33 depicts one of the three queries executed through the AccessBIM framework.

The first query executed through the AccessBIM framework retrieved 2891 rows within 0.202s, which represented 484 KB (0.47 MB) of data. This was then compared to the outcome of Mithani's paper which retrieved 0.5 MB of records within 0.75 s.

The second query executed through the AccessBIM framework retrieved 4823 rows within 0.412s, which represented 871 KB (0.85 MB) of data while Mithani’s solution has taken 1.25s to retrieve 0.9 MB of records.

The third query executed through the AccessBIM framework retrieved 13,482 rows within 1.003s, which represented 1652 KB (1.61 MB) of data while Mithani’s solution has taken 1.75s to retrieve 1.6 MB of records.

Table 6.7 summarizes the results obtained by executing the three queries through the AccessBIM framework in comparison to the results obtained in Mithani’s paper.

Table 6.7: Comparison between the solution described in the referenced paper and AccessBIM framework

Solution described in the referenced research		AccessBIM framework	
Size of data retrieved (MB)	Execution time (s)	Size of data retrieved (MB)	Execution time (s)
0.5	0.75	0.47	0.202
0.9	1.25	0.85	0.412
1.6	1.75	1.61	1.003

Figure 6.34 depicts the comparison between the referenced research and the AccessBIM framework. The y-axis represents the query execution time in seconds while the x-axis represents the size of the data retrieved in Megabytes. It is observed that there is a significant difference in the execution times of the AccessBIM framework and the solution described in the referenced research. The queries executed via the AccessBIM framework has experienced a lesser execution time compared to the queries executed via the solution described in the referenced research despite of the similar size of data retrieved.

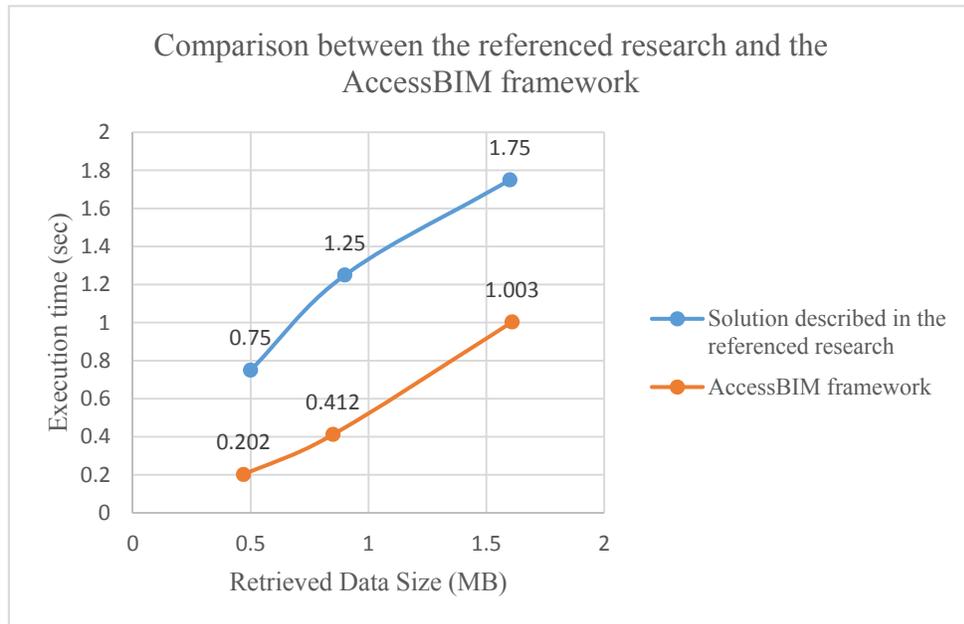


Figure 6.34: Comparison between the solution described in the referenced paper and AccessBIM framework

Hence, it can be concluded that the AccessBIM framework has better performance in terms of database optimization. This is due to the fact that the AccessBIM framework utilizes three database optimization techniques in addition to the four algorithms described in chapter 4, while the solution introduced in the reference paper utilizes a single algorithm, depicted in figure 6.35, in addition to join and aggregation, query rewriting and an alternate query with execution plan for query optimization.

<p>Proposed Algorithm</p> <p>Algorithm QOPT(Input Query)</p> <p>Description: Input Query which is used to retrieve data from given dataset, proposed algorithm provide an optimized execution plan at minimum cost, by identifying the missing term and perform required operation.</p> <p>Input : Query.</p> <p>Output : Retrieve data with minimum cost (with an optimized evaluation plan).</p> <p>1: Scan table & schema. Calculate the query execution cost for current execution plan.</p> <p>2: Design the initial tree related to query.</p> <p>3: Move the select operation down the query tree.</p> <p>4: Rewrite the query. (By finding the missing index, multi table join, redundantly used table, Reanalyze the schema etc.)</p> <p>5: Execute the query which gives the best evaluation plan with minimum cost.</p>
--

Figure 6.35: Proposed algorithm of the referenced paper (Taken from: ‘A Novel Approach for SQL Query Optimization’ [79])

6.5.2 Comparison between the frequency of executing the query and the number of users

J. A. D. C. A. Jayakody et al.’s paper on “A Database Optimization Model with Quantitative Benchmark” [80] proposed a database optimization model that is capable of significantly reducing the amount of query execution time thus improving the optimization efficiency. The research utilized 1500 records from a blood information management system to validate the efficiency and effectiveness of the proposed optimization model. Results were obtained for queries executed simultaneously by the database with the increasing number of users. The optimization model has been tested and simulated using a MYSQL database run on machine that uses an Intel® Core™ i7 processor and a 8GB RAM.

In order to assess the degree of performance of the AccessBIM framework, the simulation was run on a windows machine that used an Intel® Core™ i7 processor, and a query similar to the one used in the research paper was executed using the PostgreSQL AccessBIM database which contained 1555 records.

Although the referenced research paper utilized different queries on a different system, its results could be used to evaluate the performance of the AccessBIM framework as both models have been implemented in a machine with identical technical specification. Besides, both the frameworks utilized a phpMyAdmin graphical user interface as the simulation software.

```

SET profiling = 1;
SELECT donors.name, products.name, donations.quantity
FROM donors, products, donations
INNER JOIN donations ON donors.id =
donations.donor_id
INNER JOIN products ON products.id =
donations.product_id WHERE products.category_id = 1;
SHOW profiles;

```

Figure 6.36: Query used in the referenced paper (Taken from: A database optimization model with quantitative benchmark [80])

Figure 6.37 depicts the query executed to test the performance of the AccessBIM database while figure 6.36 illustrates the query used in the referenced research paper. It is observed that both queries are similar to a greater extent as they contain an INNER JOIN with an ‘ON’ clause.

```

Simulation_Based_Indoor_Navigation_System on postgres@PostgreSQL 9.6
1  select distinct movement_info.x_coordinate,
2      movement_info.y_coordinate,
3      path_history_info.user_id_fk
4      from movement_info
5          INNER JOIN path_history_info
6  ON movement_info.path_history_id_fk=path_history_info.path_history_id
7      where path_history_id_fk in
8  (select max(path_history_id)
9   from path_history_info
10      group by user_id_fk);
11

```

Figure 6.37: Query executed by the AccessBIM framework

After executing the query, its frequency of execution was determined to obtain a considerable difference as the time of execution takes a very small value. The frequency of executing the query was calculated as follows:

$$\text{Frequency of execution} = \frac{1}{\text{Execution time}} \tag{Equation 9}$$

Table 6.8 compares the frequency of execution for the optimization model proposed in the paper with the frequency of execution of the AccessBIM framework for the increasing number of users.

Table 6.8: Frequency of execution with changing number of users

Number of users (n)	Frequency of execution of the optimization model in the referenced paper	Frequency of execution of the AccessBIM framework
1	2074.69	2080.81
2	1219.51	1517.08
3	766.28	1477.54
4	634.12	1432.17
5	558.66	1414.00
6	508.13	1395.20
7	422.48	1361.43
8	382.36	1330.54
9	343.64	1280.83
10	324.46	1240.93

Figure 6.38 compares the frequency of execution of the AccessBIM model with the optimization model proposed in the referenced research.

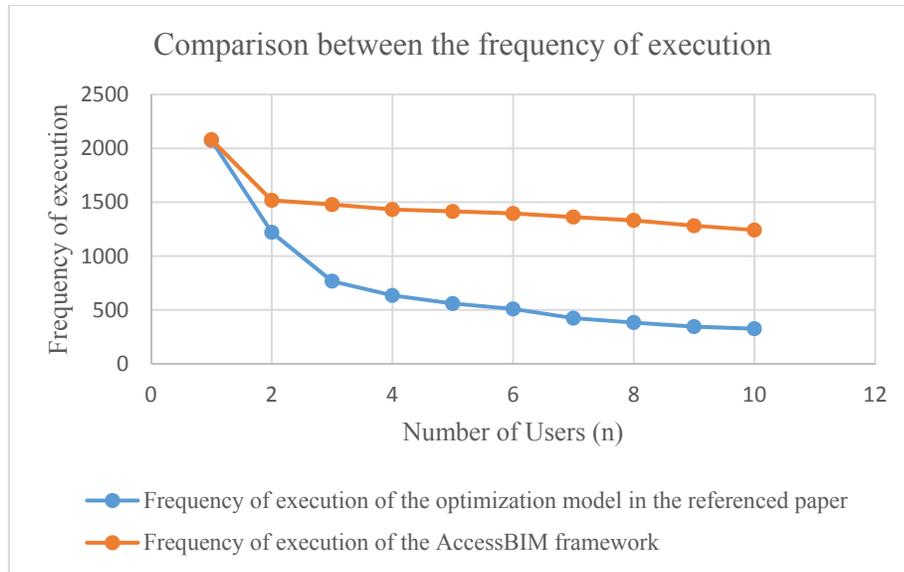


Figure 6.38: Comparison between the frequency of execution

It is observed that frequency of execution and the number of users have an inversely proportional relationship where the frequency of execution approaches zero when the number of users approaches infinity. The frequency of execution of the AccessBIM framework decreases by a lesser amount when compared to the database optimization model of the referenced paper [80]. This indicates that more users can request navigation assistance simultaneously from the AccessBIM framework rather than the optimization model suggested in the referenced paper.

6.6 Chapter conclusion

Real-time map generation for indoor navigation is a tough task to achieve due to the rapid environmental changes. The AccessBIM framework is optimized using stored procedures and query rewriting. BIMcache, which is a time reduction mechanism is also applied to optimize the AccessBIM framework. When navigators collect and submit crowdsourced data into the optimized framework, the real-time map generated within a few seconds to facilitate vision-impaired individuals.

Since the crowdsourced data for this study was collected via a simulated environment, there could be changes in the results when the framework is applied in real world due to

uncontrollable environmental factors such as alterations in mobile device platforms, varying strength of Wi-Fi signals, bandwidth issues and accuracy fluctuations in mobile sensors. Furthermore there would be latency issues in getting data from the user to the database in an actual environment.

Following relationships shows the correlation between stored procedure, query rewriting, BIMcache reduction time and database optimization.

{Stored procedures, Query rewriting} \in Database optimization

{Database optimization, BIMcache time reduction} \in Optimized AccessBIM

{Optimized AccessBIM, Crowded sourced data} \longrightarrow Real-time map

This chapter explains the evaluation matrices, query execution time and query processing time taken as the baseline to assess the performance of the AccessBIM framework.

Further, this chapter illustrates the database optimization layer which was discussed in section 3.3.1, figure 3.2 with the test results of data insertion and the retrieval involved to generate a real-time map, which was discussed graphically. A quantitative benchmark was designed to evaluate the performance enhancement of the AccessBIM framework.

Chapter 7 – Conclusion and Future Research

7.1 Chapter Overview

This chapter discusses the findings of the research with regard to the problem statement and the general conclusions that were derived from the findings. Section 7.2 summarizes the findings of the research and its significance while section 7.3 introduces a recommended framework for database optimization and real-time map generation that could be implemented in similar research work. Section 7.4 discusses the limitations of the research while section 7.5 refers to the areas of future work.

7.2 Research outcomes and significance

The foremost objective of this research was to determine:

“How indoor structural characteristics could be stored, optimized and used to generate a real-time map of an indoor environment?”

With the aim of addressing the above question, the author wished to improve the quality of life for people with vision impairment by assisting them in their indoor navigation, by the generation of a real-time indoor map that used an improved schema design which incorporated database optimization techniques such as stored procedures, indexing, query rewriting and BIMcache. Thus, the AccessBIM model was developed to ease real-time indoor map generation with enhanced performance and speed, to assist vision impaired in their navigation.

The following significant outcomes were achieved through the research:

- A method to store and traverse data within the AccessBIM server and retrieve related features and thus generate a map of the spatial environment using the received simulated crowdsourced data.
- Introduced a quantitative database optimization benchmark.
- Represent building information within the model to be used by vision impaired individuals to help navigation and path finding in an unknown environment.

- Ensures multi-user access of the AccessBIM, due to the real-time nature. In a real environment, multiple users can access the AccessBIM server and receive related information, features, and updates that are specific to the place.
- Allow correction and changes in floor plans when errors or replacements are encountered.

This thesis introduced the AccessBIM framework of environmental characteristics that store and generate an accurate map based on the real-time indoor environmental changes. The AccessBIM framework tested using a simulator, eases accessibility of the vision impaired as it aids the management of indoor spatial data in a digital format to assist vision impaired individuals to independently access unfamiliar indoor environments via the generation of an indoor map in real-time. The AccessBIM framework collects simulated crowdsourced data through an API, thus saving time as collecting data via crowdsourcing is much faster compared to collecting data via mobile robots. The use of the AccessBIM framework also ensures less confusion as the map is being updated regularly with real-time changes of the indoor environment. Furthermore, the use of database optimization techniques speeds up the process of query execution, enabling the map to be generated as and when requested by a user.

The AccessBIM database is capable of achieving the above benefits due to faster query processing, lesser cost per query and efficient use of the database engine due to less memory consumption. The author created 32 stored procedures to ensure speed of high end data transactions and to minimize round trips between the AccessBIM database and the application to reduce the execution time in generating a real-time map. As the research was based on simulated crowdsourced data, indexing was applied only to 8 stored procedures that were critical in minimizing the execution time while indexing was not applied to the rest of the stored procedures. The use of indexing on stored procedures enhanced the speed of the database to a greater extent.

Similarly, a memory caching mechanism known as BIMcache was utilized for database optimization. The BIMcache facilitates in reducing the time taken for map generation by searching the database for already existing movement information, thus saving time and the cost of searching data. The use of stored procedures, indexing and BIMcache would optimize

the performance of any database, thus enabling the insertion and retrieval of information in real-time through faster query execution.

The AccessBIM database is also equipped with four algorithms that correlate the AccessBIM framework with other indoor navigation systems. The Crowdsourced Data Collection Algorithm identifies the location of each user and obstacle on a periodic basis, in the form of X and Y coordinates while the Database Optimizer algorithm applies stored procedures and query rewriting to the relations in the database. The BIMcache Optimizer Algorithm acts similar to a caching mechanism by searching the database for already existing paths thus saving time and the cost of searching data. The synthesis of these three algorithms facilitates the Real-Time Map Generation Algorithm to generate a meaningful indoor map with minimal consumption of resources.

The use of database optimization makes it convenient to search the required data from large data sets while facilitating the identification of the best query for retrieving data using minimal resources. This enables high performance of the system via faster processing and lesser database cost thereby increasing the performance of the real-time map generation layer which leads to efficient use of the database engine.

Table 7.1 summarizes the results of the three database optimization techniques used in the research.

Table 7.1: Summary of research results

Database optimization technique	Results	
	Without optimization	With optimization
Stored procedures & indexing	23.077 s	4.22 s
BIMcache	16.994 s	9.335 s
Query rewriting	Retrieved 541 rows	Retrieved 170 rows

The total time taken for map generation depends on the time taken by a user to navigate within the simulator, the time taken to process the collected data and the time taken to generate the real-time map based on collected data. Thus, figure 7.1 summarizes the difference in query execution time and the total map generation time with and without database optimization when five users request navigation assistance simultaneously.

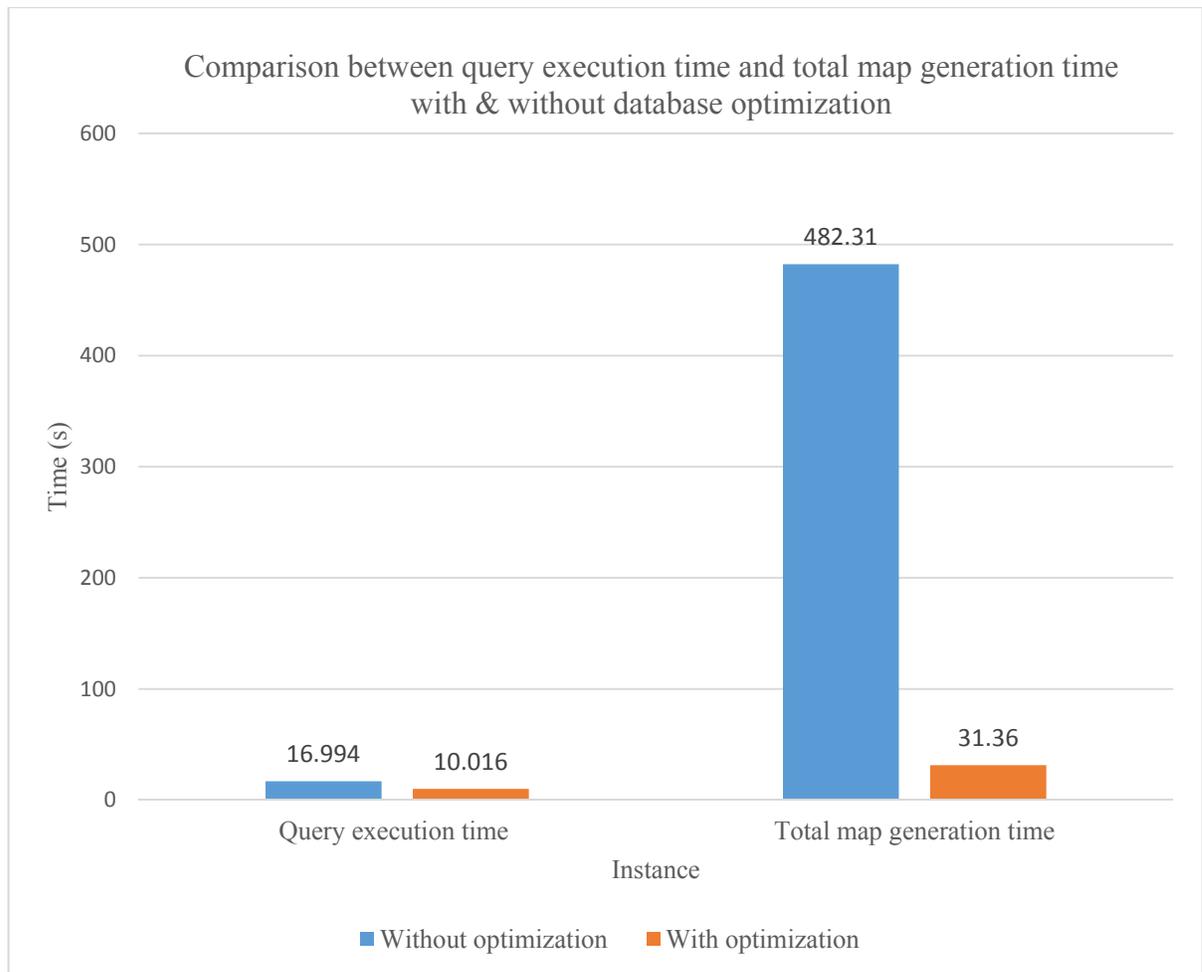


Figure 7.1: Comparison between query execution time and total map generation time with & without database optimization

It is evident that the synthesis of the three database optimization techniques has significantly reduced the total map generation time facilitating the indoor map to be generated in real-time. Besides, the synthesis of the revised database optimization techniques together with the map construction algorithms introduced in the research is competent to be implemented in any other database offering indoor and outdoor navigation assistance to not only the vision

impaired individuals but also tourists and the aged who are unfamiliar with complex and dynamic environments.

The AccessBIM framework turns out to be the author's foremost research contribution as the entire study is based on it while database optimization and the real-time map generation algorithm also stands out to be significant contributions to the research community. Hence, it is understood that the author has made three major contributions to the research community related to vision impaired indoor navigation and database optimization.

According to Helen Keller, "What a blind person needs is not a teacher but another self", thus the AccessBIM model of environmental characteristics for vision impaired indoor navigation and wayfinding stands in place of another self by generating an indoor map in real-time with the aid of algorithms and the fusion of multiple database optimization techniques introduced in the research. Hence, the author's contribution of how crowdsourced data can be used to create a real-time map that facilitates vision impaired indoor navigation in known and unknown environments was clearly demonstrated using the AccessBIM model of environmental characteristics.

7.3 Recommended framework

Based on the significant outcomes of the research, the author presents a recommended framework for database optimization and real-time map generation which could be implemented in any other indoor or outdoor navigation related research work. The recommended database optimization model is capable of being employed in any database to minimize the time of query execution irrespective of its resolution and characteristics. Similarly, the recommended real-time map generation model is proficient to be implemented in any research related to providing indoor and outdoor navigation assistance to individuals through the generation of a real-time map.

7.3.1 Recommended database optimization model

The recommended database optimization model illustrated in figure 7.2 provides an optimized execution plan at minimum cost, by identifying the missing terms in the input query relations, to retrieve data from a dataset.

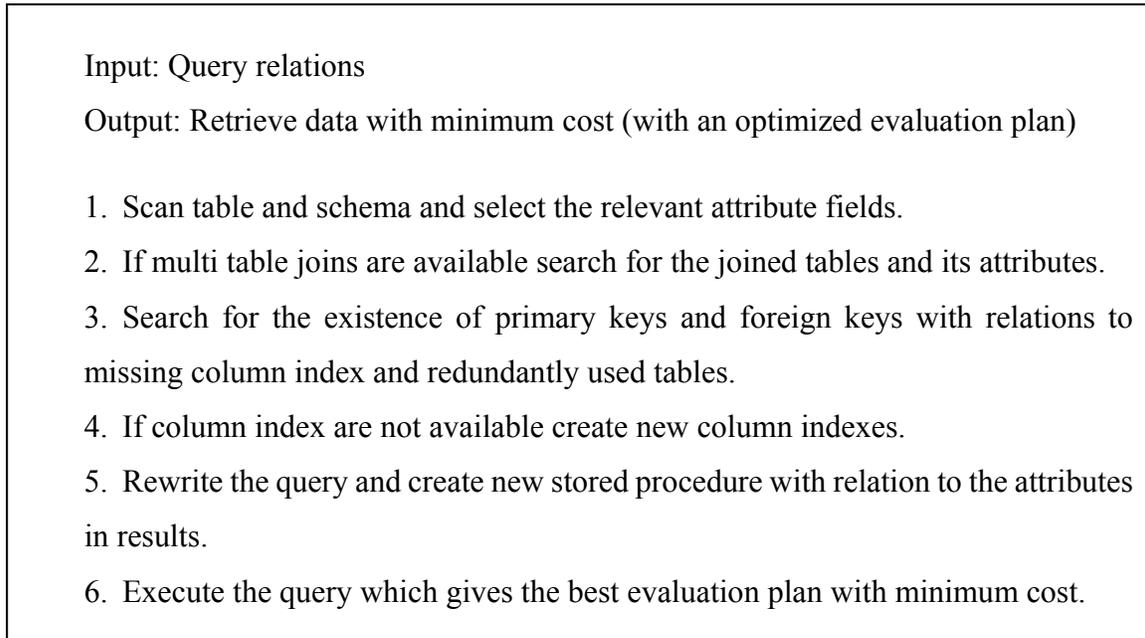


Figure 7.2: Recommended database optimization model

7.3.2 Recommended real-time map generation model

The recommended real-time map generation model illustrated in figure 7.3 generates the real-time map based on collected and optimized crowdsourced data and stores the optimal path in AccessBIM cache memory 'BIMcache'

Input: Request for generating a real-time indoor map

Output: Dynamic real time map

1. When a user requests for guidance to a destination in an indoor environment, first check BIMCACHE for existing map with optimal path
2. If a path already exists draw the dynamic real time indoor map with optimized data
3. If a map is not available in BIMcache, collect environmental characteristics on users and objects and store in AccessBIM database
4. Query the database for the path information with the crowdsourced data input.
5. Load map and update BIMCACHE with new optimized environmental path information
6. Repeat the above steps until the user reaches the destination

Figure 7.3: Recommended real-time map generation model

7.4 Limitations of the research

As the work presented in this thesis is a part of a large research project that focuses on developing an indoor navigation system for vision impaired individuals that uses smartphones and tablets, few unavoidable limitations prevailed. Since the crowdsourced data for this study was collected via a simulated environment, there can be changes in results when the framework is applied in real world due to uncontrollable environmental factors such as alterations in mobile device platforms, varying strength of Wi-Fi signals and accuracy fluctuations in mobile sensors. However, there will be no direct impact on the AccessBIM model as the simulated data used in the testing were evaluated using a database optimization benchmark.

7.5 Future Research Work

While this thesis demonstrated the potential of AccessBIM model by collecting simulated crowdsourced data in an indoor environment, many opportunities for extending the scope of this thesis still remains. Hence, the following ideas could be suggested as future extensions of the research.

A voice-user interface could be integrated with the AccessBIM model to enhance human computer interaction where the system is queried with voice based instructions, thus making navigation even more convenient for vision impaired individuals. Similarly, the AccessBIM model could be integrated with existing building management systems to provide real-time information access to the environmental characteristics of specific buildings. The model also requires a security framework to maintain integrity of the collected environmental information.

Furthermore, the use of idempotent transactions and idempotence as an optimization method would allow extensive query caching. As per this approach two servers could be used in real-time map generation where the first server commits crowdsourced information about a set of commands to indicate that the set has committed whereas the second server determines whether the user's request identified the latest set received for map generation in a corresponding session.

Thus, the use of this approach would enable extensive query caching in instances which cannot be cached easily.

References

- [1] World Health Organization, "World Health Organization," October 2017. [Online]. Available: <http://www.who.int/mediacentre/factsheets/fs282/en/>. [Accessed 3 January 2018].

- [2] World Health Organization, "Lions Center for the Visually Impaired," 2016. [Online]. Available: <http://www.seniorvision.org/resources/facts-about-blindness-and-visual-impairment>. [Accessed 5 April 2018].

- [3] University of Hertfordshire, "Visual Impairment: Its Effect on Cognitive Development and Behaviour," [Online]. Available: <http://www.intellectualdisability.info/physical-health/articles/visual-impairment-its-effect-on-cognitive-development-and-behaviour>. [Accessed 10 Apr 2017].

- [4] learn.org, "What Is CAD Drafting?," [Online]. Available: http://learn.org/articles/What_is_CAD_Drafting.html. [Accessed 17 Apr 2017].

- [5] "Potential Problems with GPS Tracking," Auto Alert Limited, 2013. [Online]. Available: <http://www.autoalert.me.uk/problems-with-gps-tracking/>. [Accessed 02 Dec 2016].

- [6] S.Bhandari, "Reasons for Slow Database Performance," 30 Apr 2011 . [Online]. Available: <https://dzone.com/articles/reasons-slow-database>. [Accessed 20 Apr 2017].

- [7] M. Petrovic, "SQL Shack," 14 April 2014. [Online]. Available: <https://www.sqlshack.com/poor-database-indexing-sql-query-performance-killer-recommendations/>. [Accessed 04 January 2018].
- [8] Julius, "Electronic Mobility Devices for Persons Who are Blind or Visually Impaired," 12 Sep 2010. [Online]. Available: <http://evengrounds.com/blog/electronic-mobility-devices-for-persons-who-are-blind-or-visually-impaired>. [Accessed 28 Dec 2016].
- [9] A. Leyden, "40 Uses For Smartphones in School," 19 Feb 2015. [Online]. Available: <https://www.goconqr.com/en/examtime/blog/40-uses-for-smartphones-in-school/>. [Accessed 09 Dec 2016].
- [10] A. Ilindra, "How has Smartphones Made Our Life Easier?," 26 Nov 2016. [Online]. Available: <http://www.geekdashboard.com/smartphone-made-our-life-easier/>. [Accessed 09 Dec 2016].
- [11] M. Milošević, M. T. Shrov and E. Jovanov, "Applications of Smartphones for Ubiquitous Health Monitoring and Wellbeing Management," *Journal of information technology and applications*, vol. 1, no. 1, pp. 7-15, 2011.
- [12] MiTAC International, "What is GPS?," [Online]. Available: <http://www.mio.com/technology-what-is-gps.htm>. [Accessed 04 Jan 2017].
- [13] Laboratory, University of Maryland Space Systems, "Inertial Measurement Unit (IMU)," 2013. [Online]. Available:

<http://www.ssl.umd.edu/projects/RangerNBV/thesis/2-4-1.htm>. [Accessed 09 Dec 2016].

- [14] J. Victor and et-al, "Indoor navigation with smart phone IMU for the visually impaired in university buildings," *Journal of Assistive Technologies*, vol. 10, no. 3, pp. 133 - 139, 2016.
- [15] C. Wen, S. Pan, C. Wang and J. Li, "An Indoor Backpack System for 2-D and 3-D Mapping of Building Interiors," in *IEEE Geoscience and Remote Sensing Letters*, 2016.
- [16] J. Tang and Y. Chen, "Fast Fingerprint Database Maintenance for Indoor Positioning Based on UGV SLAM," in *Special Issue Sensors for Indoor Mapping and Navigation*, 2015.
- [17] "merriam-webster," 3 December 2017. [Online]. Available: <https://www.merriam-webster.com/dictionary/crowdsourcing>. [Accessed 3 January 2018].
- [18] Statista, "Number of mobile phone users worldwide from 2013 to 2019 (in billions)," Aug 2015. [Online]. Available: <https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/>. [Accessed 09 Dec 2016].
- [19] N. Abhayasinghe and I. Murray, "A novel approach for indoor localization using human gait analysis with gyroscopic data," in *International conference on indoor positioning and indoor navigation*, Sydney, AU, 2012.

- [20] N.Rajakaruna and I.Murray, "Efficient and Adaptive Generic Object Detection Method for Indoor Navigation," in *International conference on indoor positioning and indoor navigation*, Montbeliard,belfort,Fr, 2013.
- [21] A.Riazi,C.Bridge, "Potential environmental hazard from perspectives of people with central vision loss who reside in Sydney," *Independent Living Journal*, vol. 29, no. 1, pp. 16-21, 2013.
- [22] R. G. Golledge, J.R. Marston, J.M. Loomis,R. L. Klatzky, "Stated Preferences for Components of a Personal Guidance System for Nonvisual Navigation," *Journal of Visual Impairment & Blindness*, vol. 98, no. 3, pp. 135-147, 2014.
- [23] Unite For Sight, " Eye Disease and Mental Health," [Online]. Available: http://www.uniteforsight.org/community-eye-health-course/module11#_ftn1. [Accessed 20 Dec 2016].
- [24] Dr.M.Saarela, "Solving way-finding challenges of a visually impaired person in a shopping mall by strengthening landmarks recognisability with iBeacons," Finland, 2015.
- [25] Abbas Riazi, Fatemeh Riazi,Rezvan Yoosfi,Fatemeh Bahmeei, "Outdoor difficulties experienced by a group of visually impaired Iranian people," *Current Ophthalmology*, vol. 28, no. 2, pp. 85-90, 2016.
- [26] Society for Accessible Travel and Hospitality, "How to travel with sight impairment or blindness," 123YourWeb.com, [Online]. Available: <http://sath.org/how-to-travel-with-a-sight-impairment-or-blindness>. [Accessed 18 Dec 2016].

- [27] Texas School for the Blind and Visually Impaired, "Specific Eye Conditions, Corresponding Impact on Vision, And Related Educational Considerations," [Online]. Available: <http://www.tsbvi.edu/eye-conditions>. [Accessed 20 Dec 2016].
- [28] W. Elmannai and K. Elleithy, "Sensor-Based Assistive Devices for Visually-Impaired People: Current Status, Challenges, and Future Directions," *Sensors*, vol. 17, no. 3, 2017.
- [29] P. Chanana, R. Paul, M. Balakrishnan and P. V. Rao, "Assistive technology solutions for aiding travel of pedestrians with visual impairment," *Journal of Rehabilitation and Assistive Technologies Engineering*, vol. 4, pp. 1-16, 2017.
- [30] B. HITZ, "Photonics.com," Photonics media, June 2003. [Online]. Available: https://www.photonics.com/a16107/Lasers_Assist_the_Blind. [Accessed 5 May 2018].
- [31] K. Elleithy and W. Elmannai, "Sensor-Based Assistive Devices for Visually-Impaired People: Current Status, Challenges, and Future Directions," *Sensors*, vol. 17, no. 3, 2017.
- [32] Able Data, "Able Data," 19 April 2012. [Online]. Available: <https://abledata.acl.gov/product/sonic-pathfinder>. [Accessed 5 May 2018].
- [33] The Miniguide mobility aid, "The Miniguide mobility aid," [Online]. Available: http://www.gdp-research.com.au/minig_1.htm. [Accessed 5 May 2018].

- [34] Assistive IT, "Assistive IT," [Online]. Available: <http://www.assistiveit.co.uk/VI-Products/Portable-Devices/Trekker-Breeze-GPS>. [Accessed 4 May 2018].
- [35] H. Liu, H. Darabi, P. Banerjee and J. Liu, "Survey of Wireless Indoor Positioning Techniques and Systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 6, pp. 1067-1080, 2007.
- [36] S. S. Saab and Z. S. Nakad, "A Standalone RFID Indoor Positioning System Using Passive Tags," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 5, pp. 1961-1970, 2011.
- [37] J. Yang, . Z. Wang and X. Zhang, "An iBeacon-based Indoor Positioning Systems for Hospitals," *International Journal of Smart Home*, vol. 9, no. 7, pp. 161-168, 2015.
- [38] O.Lahav, "Virtual reality as orientation and mobility aid for blind people," *Journal of Assistive Technologies*, vol. 8, no. 2, pp. 95 - 107, 2014.
- [39] American Foundation for the Blind, "Orientation and Mobility Skills," [Online]. Available: <http://www.visionaware.org/info/everyday-living/essential-skills/an-introduction-to-orientation-and-mobility-skills/123>. [Accessed 22 Dec 2016].
- [40] Y.W. Weidong and C.J Wang, "Map-based localization for mobile robots in high-occluded and dynamic environments," *Industrial Robot: An International Journal*, vol. 41, no. 3, pp. 241 - 252, 2014.

- [41] A. Beamer, "Map metadata: essential elements for search and storage," *Program*, vol. 43, no. 1, pp. 18 - 35, 2008.
- [42] M.Tanaka,Y.Mizuchi,A.Suzuki,H.Imamura,Y.Hagiwara, "Enhanced view-based navigation for human navigation by mobile robots using front and rear vision sensors," in *International conference on indoor positioning and indoor navigation*, Montbéliard, France, 2013.
- [43] C. Ching and J. Bianca, "Mobile Indoor Positioning Using Wi-Fi Localization and Image Processing," in *Proceedings in Information and Communications Technology*, 2012.
- [44] P. Benavidez, M. Muppidi, P. Rad, J. J. Prevost, M. Jamshidi and L. Brown, "Cloud-based realtime robotic Visual SLAM," in *2015 Annual IEEE Systems Conference (SysCon) Proceedings*, Vancouver, BC, 2015.
- [45] J. Dhruv, A. Prabhav and M. Aman, "DESIGN AND USER TESTING OF AN AFFORDABLE CELL-PHONE BASED INDOOR NAVIGATION SYSTEM FOR VISUALLY IMPAIRED," New Delhi, 2012.
- [46] RF Wireless World, "RF Wireless World," RF Wireless World, [Online]. Available: <http://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-Infrared-Sensor.html>. [Accessed 12 March 2018].
- [47] Y. Wu, W. Guo, C.-Y. Chan and K.-L. Tan, "Fast Failure Recovery for Main-Memory DBMSs on Multicores," in *SIGMOD '17- Proceedings of the 2017 ACM International Conference on Management of Data*, Chicago, 2017.

- [48] "TechTarget," July 2006. [Online]. Available: <http://searchsqlserver.techtarget.com/definition/ACID>. [Accessed 3 January 2018].
- [49] "Data Source Consulting," Data Source Consulting, 25 April 2015. [Online]. Available: <http://ds.datasourceconsulting.com/blog/data-profiling/>. [Accessed 3 January 2018].
- [50] F. Yuanyuan and M. Xifeng, "Distributed database system query optimization algorithm research," in *2010 3rd International Conference on Computer Science and Information Technology*, Chengdu, 2010.
- [51] "DB-Engines," solid IT gmbh, 01 Oct 2012. [Online]. Available: http://db-engines.com/en/ranking_trend. [Accessed 25 Nov 2016].
- [52] "PostgreSQL," The PostgreSQL Global Development Group, [Online]. Available: <https://www.postgresql.org/about/advantages/>. [Accessed 10 Dec 2016].
- [53] D.Thakur, "What is Object-Relational Database Systems? Advantages and Disadvantages of ORDBMSS.," [Online]. Available: <http://ecomputernotes.com/database-system/adv-database/object-relational-database-systems>. [Accessed 25 Feb 2017].
- [54] M.Wang, "Implementation of Object-Relational DBMSs in a Relational Database Course," in *Special Interest Group on Computer Science Education*, Charlotte, NC USA , 2001.

- [55] P.Karthik, G.T.Reddy and E.K.Vanan, "Tuning the SQL Query in order to Reduce Time Consumption," *IJCSI International Journal of Computer Science*, vol. 9, no. 4, pp. 418-423, 2012.
- [56] L. Hong, M. Lu and W. Hong, "A Business Computing System Optimization Research on the Efficiency of Database Queries," in *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, Beijing,CN, 2013.
- [57] I.Jimenez,J. LeFevre, N.Polyzotis, H.Sanchez. and K. Schnaitter, "Benchmarking Online Index-Tuning Algorithms," *IEEE Data Eng. Bull.*, vol. 34, no. 4, pp. 28-35, 2011.
- [58] S. W. Schlosser and S. Iren, "Database storage management with object-based storage devices," in *Workshop on Data Management on New Hardware*, Maryland, 2005.
- [59] O. Papaemmanouil, M. Cherniack and Z. Li, "OptMark: A Toolkit for Benchmarking Query Optimizers", in *The 25th ACM International on Information and Knowledge Management*, Indianapolis, 2016.
- [60] N. Roussopoulos, "View Indexing in Relational Databases," *ACM Transactions on Database Systems (TODS)*, vol. 7, no. 2, pp. 258-290, 1982.
- [61] J. A. D. C. A. Jayakody and I. Murray, "The construction of an indoor floor plan using a smartphone for future usage of blind indoor navigation," in *2014*

International Conference on Contemporary Computing and Informatics (IC3I), Mysore, 2014.

- [62] J.A.D.C Jayakody and I. Murray, "Proposed novel schema design for map generation to assist vision impaired in an indoor navigation environment," in *2014 International Conference on Contemporary Computing and Informatics (IC3I)*, Mysore, 2014.

- [63] M. Richards, "Software Architecture Patterns," Safari Books Online, [Online]. Available: <https://www.safaribooksonline.com/library/view/software-architecture-patterns/9781491971437/ch01.html>. [Accessed 8 May 2018].

- [64] J. Maymala, *PostgreSQL for Data Architects*, India: Packt Publishers, 2015.

- [65] Raghavendra, "Relational Database Technologies," July 2011. [Online]. Available: <http://raghavt.blogspot.com/2011/07/pgmemcache-setup-and-usage.html>. [Accessed 11 November 2016].

- [66] Essential SQL, "Database Indexes Explained," Essential SQL, [Online]. Available: <https://www.essentialsql.com/what-is-a-database-index/>. [Accessed 15 May 2018].

- [67] Heroku Dev Centre, "Efficient Use of PostgreSQL Indexes," Heroku Dev Centre, [Online]. Available: <https://devcenter.heroku.com/articles/postgresql-indexes>. [Accessed 15 May 2018].

- [68] PostgreSQL, "PostgreSQL," PostgreSQL, [Online]. Available: <https://www.postgresql.org/docs/8.4/static/indexes-expressional.html>. [Accessed 15 May 2018].
- [69] R.McHaney, Understanding Computer Simulation, Denmark : BookBoon 2009, 2009.
- [70] C. Qiu and M. W. Mutka, "iFrame: Dynamic indoor map construction through automatic mobile sensing," in *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Sydney, NSW, 2016.
- [71] A. Jayakody, Director, *Test Case 1*. [Film]. Sri Lanka.2017.
- [72] A. Jayakody, Director, *Test Case 2*. [Film]. Sri Lanka.2017.
- [73] A. Jayakody, Director, *Test Case 3*. [Film]. Sri Lanka.2017.
- [74] A. Jayakody, Director, *Test Case 4*. [Film]. Sri Lanka.2017.
- [75] A. Jayakody, Director, *Test Case 5*. [Film]. Sri Lanka.2017.
- [76] N. Ritter and W. Zhang, "The Real Benefits of Object-Relational DB-Technology," in *18th British National Conference on Databases*, Germany, 2001.
- [77] J.H.Hanke,D.W.Wichern, Business Forecasting, New Delhi: Prentice hall of India, 2007.

- [78] PostgreSQL Tutorial , "PostgreSQL Stored Procedures," [Online]. Available: <http://www.postgresqltutorial.com/postgresql-stored-procedures/>. [Accessed 05 Feb 2017].
- [79] F. Mithani, S. Machchhar and F. Jasdanwala, "A Novel Approach for SQL Query Optimization," in *IEEE International Conference on Computational Intelligence and Computing Research (ICCCIC)*, Chennai, 2016.
- [80] J. A. D. C. A. Jayakody, S. Lokuliyana, I. Murray, J. Hermann, D. S. A. Kandawala and S. E. C. Nanayakkara, "A database optimization model with quantitative benchmark," in *International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)* , New Delhi, 2016.

Every reasonable effort has been made to acknowledge the owners of copyright material. I would be pleased to hear from any copyright owner who has been omitted or incorrectly acknowledged.

Appendices

Appendix A: System Specification

The system under test has the following features:

- Microsoft Windows 8.1, 64 bit version as the operating system.
- As real-time-database management system products, PostgreSQL windows version
- Both the operating system and the database management system run on a Hardware platform having the following features:
 - 64 bit machine
 - Intel® Core™ i5-2430M CPU @ 2.40GHZ
 - 8 GB of RAM
 - 500GB of Hard Disk

Appendix B: C# DLL to interact C# application and web service

```
//Following segment contains names of common libraries used for the implementation

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net;

using System.Text;

using System.Threading.Tasks;

//Following code segment contains set of key words used for simulation implementation

namespace INS

{

    public class Api

    {

        public void insert_object_info(string lbl_id_fk,string b_id_fk,string f_id_fk,string

image_description,string category ,string x_coordinate,string y_coordinate) {

            string post_url =

"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/insert_object_info.php";

            string POST_DATA = "lbl_id_fk=" + lbl_id_fk + "&b_id_fk=" + b_id_fk + "&f_id_fk=" + f_id_fk +

"&image_description=" + image_description + "&category=" + category + "&x_coordinate=" + x_coordinate +

"&y_coordinate=" + y_coordinate;

            string responseString = DataRequestToServer(POST_DATA, post_url);

        }

    }

}
```

```

public void update_object_info(string object_id, string b_id_fk, string f_id_fk, string x_coordinate, string
y_coordinate)

    {
string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/update_object_info.php";

string POST_DATA = "object_id=" + object_id + "&b_id_fk=" + b_id_fk + "&f_id_fk=" + f_id_fk +
"&x_coordinate=" + x_coordinate + "&y_coordinate=" + y_coordinate;

string responseString = DataRequestToServer(POST_DATA,post_url);
}
public void update_object_info_with_lable(string object_id, string lable_id, string b_id_fk, string f_id_fk,
string x_coordinate, string y_coordinate)

    { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/update_object_info_with_lable.php";

string POST_DATA = "object_id=" + object_id + "&lable_id=" + lable_id + "&b_id_fk=" + b_id_fk +
"&f_id_fk=" + f_id_fk + "&x_coordinate=" + x_coordinate + "&y_coordinate=" + y_coordinate;

string responseString = DataRequestToServer(POST_DATA, post_url);
}
public string select_registration_info(string object_id)
    {
string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/select_registration_info.php";

string POST_DATA = "object_id=" + object_id;

string responseString = DataRequestToServer(POST_DATA, post_url);

return responseString;

    }

```

```

public string insert_path_history_before_achieve(string user_id_fk_apb, string b_id_fk_apb, string
f_id_fk_apb, string starting_point_apb, string ending_point_apb)
{
    string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/insert_path_history_before_achieve.php"
;

    string POST_DATA = "user_id_fk_apb=" + user_id_fk_apb + "&b_id_fk_apb=" + b_id_fk_apb +
"&f_id_fk_apb=" + f_id_fk_apb + "&starting_point_apb=" + starting_point_apb + "&ending_point_apb=" +
ending_point_apb;

    string responseString = DataRequestToServer(POST_DATA, post_url);

    return responseString;
}
public void update_path_history_after_achieve(string path_history_id_apa, string achieve_p2_apa)
{
    string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/update_path_history_after_achieve.php";

    string POST_DATA = "path_history_id_apa=" + path_history_id_apa + "&achieve_p2_apa=" +
achieve_p2_apa;

    string responseString = DataRequestToServer(POST_DATA, post_url);
}
public string BIMcache(string starting_point, string ending_point)
{
    string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/BIMcache_movement_info.php";

    string POST_DATA = "starting_point=" + starting_point + "&ending_point=" + ending_point;

    string responseString = DataRequestToServer(POST_DATA, post_url);

    return responseString;
}
}

```

```

public string chechDoorStatus(string b_id_fk_d, string f_id_fk_d, string d_id_fk_d)
{
    string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/check_door_status.php";

    string POST_DATA = "b_id_fk_d=" + b_id_fk_d + "&f_id_fk_d=" + f_id_fk_d + "&d_id_fk_d=" +
d_id_fk_d;

    string responseString = DataRequestToServer(POST_DATA, post_url);

    return responseString;
}
public void add_movement_info(string object_id_fk_movement, string path_history_id_fk_movement, string
walking_distance_movement, string angel_movement, string direction_movement, string
time_stamp_movement, string x_coordinate_movement, string y_coordinate_movement)
{
    string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/insert_movement_info.php";

    string POST_DATA = "object_id_fk_movement=" + object_id_fk_movement +
"&path_history_id_fk_movement=" + path_history_id_fk_movement + "&walking_distance_movement=" +
walking_distance_movement + "&angel_movement=" + angel_movement + "&direction_movement=" +
direction_movement + "&time_stamp_movement=" + time_stamp_movement + "&x_coordinate_movement="
+ x_coordinate_movement + "&y_coordinate_movement=" + y_coordinate_movement;

    string responseString = DataRequestToServer(POST_DATA, post_url);
}
public void add_movement_info_with_out_object(string path_history_id_fk_movement, string
walking_distance_movement, string angel_movement, string direction_movement, string
time_stamp_movement, string x_coordinate_movement, string y_coordinate_movement)
{

```

```

string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/insert_movement_info_without_object.p
hp";

    string POST_DATA = "path_history_id_fk_movement=" + path_history_id_fk_movement +
"&walking_distance_movement=" + walking_distance_movement + "&angel_movement=" + angel_movement
+ "&direction_movement=" + direction_movement + "&time_stamp_movement=" + time_stamp_movement +
"&x_coordinate_movement=" + x_coordinate_movement + "&y_coordinate_movement=" +
y_coordinate_movement;

    string responseString = DataRequestToServer(POST_DATA, post_url); }

public string select_object_info(string building_id, string floor_id)

    { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/select_object_info.php";

    string POST_DATA = "building_id=" + building_id + "&floor_id=" + floor_id;

    string responseString = DataRequestToServer(POST_DATA,post_url);

    return responseString; }

public string select_object_info_for_specific_lable(string lable_id,string building_id, string floor_id)

    { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/select_object_info_for_specific_lable.ph
p";

    string POST_DATA = "lable_id=" + lable_id + "&building_id=" + building_id + "&floor_id=" +
floor_id;

    string responseString = DataRequestToServer(POST_DATA, post_url);

    return responseString; }

public void insert_wall_info(string f_id_fk_w, string b_id_fk_w, string starting_x, string starting_y, string
ending_x, string ending_y)

```

```

        { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/insert_wall_info.php";      string
POST_DATA = "f_id_fk_w=" + f_id_fk_w + "&b_id_fk_w=" + b_id_fk_w + "&starting_x=" + starting_x +
"&starting_y=" + starting_y + "&ending_x=" + ending_x + "&ending_y=" + ending_y;

        string responseString=DataRequestToServer(POST_DATA, post_url);
    }
public string select_wall_info(string b_id_fk_w, string f_id_fk_w)

    { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/select_wall_info.php";

        string POST_DATA = "b_id_fk_w=" + b_id_fk_w + "&f_id_fk_w=" + f_id_fk_w;

        string responseString = DataRequestToServer(POST_DATA, post_url);

        return responseString;    }

public string select_door_info(string b_id_fk_d, string f_id_fk_d)

    { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/select_door_info.php";

        string POST_DATA = "b_id_fk_d=" + b_id_fk_d + "&f_id_fk_d=" + f_id_fk_d;

        string responseString = DataRequestToServer(POST_DATA, post_url);

        return responseString;    }
public string getting_starting_and_ending(string path_history_id)

    { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/select_starting_and_ending_from_path_h
istory.php";

        string POST_DATA = "path_history_id=" + path_history_id;

        string responseString = DataRequestToServer(POST_DATA, post_url);

        return responseString; }

```

```

public string check_path_availability(string building_id,string floorId,string Spoint,string Epoint)
    { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/check_path_availability.php";

        string POST_DATA = "building_id=" + building_id + "&floorId=" + floorId + "&Spoint=" + Spoint +
"&Epoint=" + Epoint;

        string responseString = DataRequestToServer(POST_DATA, post_url);

        return responseString; }
public string select_starting_point_and_ending_point(string status)
    { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/select_starting_and_ending.php";

        string POST_DATA = "status=" + status;

        string responseString = DataRequestToServer(POST_DATA, post_url);

        return responseString; }

public string BIMcache1(string starting_point, string ending_point)
    { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/BIMcached/BIMcache_movement_info1
.php";

        string POST_DATA = "starting_point=" + starting_point + "&ending_point=" + ending_point;

        string responseString = DataRequestToServer(POST_DATA, post_url);

        return responseString; }

public string select_user_current_location(string user_id)

    { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/select_user_current_location.php";

```

```

        string POST_DATA = "user_id=" + user_id;

        string responseString = DataRequestToServer(POST_DATA, post_url);

        return responseString;    }

public string get_distinct_user_location(string user)

    { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/receive_distinct_user_location.php";

        string POST_DATA = "user=" + user;

        string responseString = DataRequestToServer(POST_DATA, post_url);

        return responseString; }

public string select_lable_info(string b_id_fk_lbl_in, string f_id_fk_lbl_in)

    { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/select_lable_info.php";

        string POST_DATA = "b_id_fk_lbl_in=" + b_id_fk_lbl_in + "&f_id_fk_lbl_in=" + f_id_fk_lbl_in;

        string responseString = DataRequestToServer(POST_DATA, post_url);

        return responseString; }

public string select_registartion_info(string user)

    { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/select_registration_info.php";

        string POST_DATA = "user=" + user;

        string responseString = DataRequestToServer(POST_DATA, post_url);

        return responseString; }

```

```

public void delete_movement_info_and_path_history_info(string status)

    { string post_url =
"http://localhost:8080/Simulation_Based_Indoor_Navigation_System/delete_movement_and_path_history_inf
o.php";

    string POST_DATA = "status=" + status;

    string responseString = DataRequestToServer(POST_DATA, post_url); }

public string DataRequestToServer(string postData, string url)

{    try

//Following code segment contains common method that used for HTTP POST request

    { HttpRequest httpWReq = (HttpRequest)WebRequest.Create(url);

        ASCIIEncoding encoding = new ASCIIEncoding();

        byte[] data = encoding.GetBytes(postData);

        httpWReq.Method = "POST";

        httpWReq.ContentType = "application/x-www-form-urlencoded";

        httpWReq.ContentLength = data.Length;

        using (Stream stream = httpWReq.GetRequestStream())

        {    stream.Write(data, 0, data.Length); }

        HttpResponse response = (HttpResponse)httpWReq.GetResponse();

        string responseString = new StreamReader(response.GetResponseStream()).ReadToEnd();

        return responseString;
    }
    catch (Exception e)

    {    return e.Message;

    } } }

```

Appendix C: 7th floor of Sri Lanka Institute of Information Technology



Figure 1: Main entrance of 7th floor

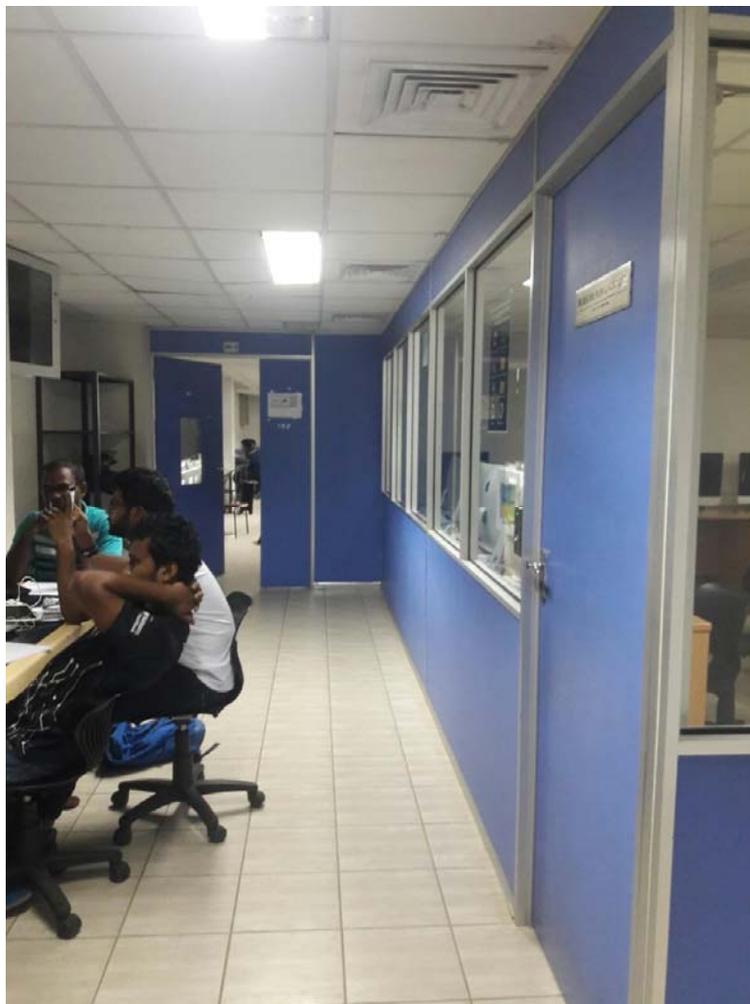


Figure 2: Paths to multimedia lab and Lab 701



Figure 3: Path to washroom 1, Maintenance room, Cabin 1, Room1 and Room2



Figure 4: Entrance of cabin 1



Figure 5: Path to washroom 2

Appendix D: Results of query execution

Test results for the comparison between number of occurrences and execution time without optimization

Number_of_users (n)	Execution_time (s) (T_{q_exe})
1	15.926
2	17.605
3	19.459
4	17.918
5	16.994
6	11.506
7	15.314
8	17.138
9	14.866
10	14.619
11	11.213
12	17.625
13	14.724
14	16.253
15	13.354
16	17.562
17	16.046
18	17.007
19	17.966
20	16.957

21	15.207
22	14.326
23	18.887
24	19.484
25	14.352
26	16.488
27	17.518
28	19.228
29	14.369
30	16.641
31	11.303
32	17.167
33	12.016
34	17.547
35	17.322
36	11.934
37	13.756
38	14.757
39	14.01
40	17.081
41	15.954
42	17.981
43	13.275
44	15.65

45	15.964
46	13.203
47	14.172
48	11.006
49	14.079
50	17.448
51	17.127
52	19.691
53	12.539
54	18.749
55	16.992
56	16.244
57	14.077
58	12.502
59	16.379
60	11.891
61	13.254
62	18.456
63	17.379
64	14.763
65	11.699
66	14.354
67	17.868
68	15.95

69	14.057
70	12.65
71	13.687
72	12.79
73	13.8
74	12.765
75	11.478
76	13.791
77	12.255
78	13.857
79	12.572
80	15.493
81	11.77
82	11.457
83	14.153
84	11.191
85	14.453
86	16.689
87	11.728
88	13.741
89	17.445
90	19.052
91	11.122
92	19.565

93	15.784
94	15.167
95	11.492
96	17.793
97	16.098
98	14.177
99	14.539
100	14.166

Test results for the comparison between number of occurrences and execution time with optimization

Number of occurrences (n)	Execution time ($T_{q.exe}$)
1	8.0097
2	10.986
3	11.28
4	8.3745
5	10.016
6	9.89
7	12.242
8	11.841
9	10.005
10	10.816
11	8.507
12	14.392

13	11.677
14	12.365
15	11.04
16	10.3341
17	9.7437
18	10.4823
19	9.7865
20	10.8232
21	11.0684
22	10.0367
23	9.4637
24	10.5717
25	9.6783
26	9.549
27	11.4994
28	10.6902
29	9.0392
30	8.2249
31	10.4183
32	10.4992
33	8.9946
34	11.077
35	9.862
36	10.0751

37	8.9187
38	8.6673
39	7.8029
40	8.6036
41	8.1625
42	8.8937
43	8.4453
44	8.9302
45	9.0691
46	10.0969
47	9.0656
48	9.5022
49	11.2213
50	10.4029
51	6.591
52	6.985
53	10.189
54	8.19
55	8.632
56	8.224
57	6.478
58	6.525
59	7.069
60	8.394

61	11.554
62	7.518
63	6.682
64	6.375
65	9.639
66	6.57
67	6.647
68	7.096
69	6.869
70	6.604
71	6.992
72	6.575
73	7.384
74	7.45
75	8.558
76	7.625
77	6.691
78	6.909
79	8.175
80	7.414
81	7.335
82	10.993
83	8.164
84	6.722

85	7.459
86	8.135
87	6.945
88	6.694
89	10.474
90	11.38
91	8.322
92	10.324
93	6.899
94	8.655
95	8.982
96	13.519
97	7.049
98	6.993
99	8.777
100	11.707

Appendix E: Summary of the Interview conducted with the Visual Impaired of Employee Federation of Ceylon

Questions asked in the interview with “The Employers' Federation of Ceylon”.

1. Your age group	15-25	26-35	36-45	46-55
-------------------	-------	-------	-------	-------

2. How long have you suffered from vision impairment	Since birth	At the age of
--	-------------	---------------

3. Are you able to find rooms in your house without others help	Yes	No
---	-----	----

4. Do you feel anxious to navigate in unknown indoor environments	Yes	No
---	-----	----

5. Do you use any technical devices to get the assistance for indoor navigation	Yes	No
---	-----	----

If yes please provide device name

.....

6. What are your expectations on an indoor navigation system?

The above questions were asked from vision impaired individuals who get computer training at The Employers' Federation of Ceylon (EFC).

Summary of the interview answers

8 out of 10 trainee students whose age between 25-45 have been suffering from vision impairment since birth. They can identify familiar places in their house such as rooms, kitchen washrooms and etc. They are also able to find their path in a familiar indoor environment. However, they find it difficult to avoid obstacles in an indoor environment which are dynamic.

They feel uncomfortable to navigate in an unfamiliar or complex environments where they do not have any idea about physical arrangement of objects. They prefer to use the white cane when navigating in an unknown environment, however, they find it difficult to navigate with confidence using the white cane in an unfamiliar environment.

Vision impaired trainees at EFC suggest that a product to support vision impaired navigation should include a way to recognize doors, windows, objects and walls separately and provide guidance for navigation so that they can reach their preferred destination easily.

Appendix F: List of Available Stored Procedures

1. check_door_status
2. check_path_availability
3. connection
4. delete_movement_and_path_history_info
5. delete_movement_info_with_achieve
6. delete_movement_info_without_achieve
7. delete_path_history_info_with_achieve
8. delete_path_history_info_without_achieve
9. email_confirm
10. insert_door_info
11. insert_movement_info
12. insert_movement_info_without_object
13. insert_object_info
14. insert_path_history_before_achieve
15. insert_wall_info
16. BIMcache_movement_info
17. receive_distinct_user_location
18. register
19. select_door_info

20. select_lable_info
21. select_object_info
22. select_object_info_for_specific_lable
23. select_registration_info
24. select_starting_and_ending
25. select_starting_and_ending_from_path_histoty
26. select_user_current_location
27. select_wall_info
28. update_door_info
29. update_object_info
30. update_object_info_with_lable
31. update_object_info_with_return_result
32. update_path_history_after_achieve