

# DE ROTTERDAM

---

## CREATING A SMARTPHONE APPLICATION TO LOCATE COLLEAGUES



Matilde Oliveti - 4323564 | Godelief Abhilakh Missier - 1541412 | Damien Mulder - 1503154  
Dimitris Zervakis - 4312775 | Haoxiang Wu - 4325591

Project coaches: Edward Verbree, Sisi Zlatanova & Rob Poll- van Dasselaar  
TU Delft, Geomatics Synthesis Project 2014  
31-10-2014

---

## Preface

This document constitutes the Final Report of the Synthesis Project of the 2<sup>nd</sup> year in MSc Geomatics, TU Delft [*GEO 2001*]. This Synthesis Project is done in cooperation with the director, project coach and the client: the Municipality of Rotterdam. The aim of the project is to develop a working prototype application to find and display the location of a colleague in a high-rise building ('De Rotterdam'). All acquired knowledge from the first year in Geomatics is used, as well as new learning experiences and discoveries within the context of working in a team.

This report consists of the requirements, concept, analysis, literature research and the implementation performed by the team in the eight weeks set for the project.

The Final Report starts with the Executive Summary, which explains in more detail the scope of the project. Then the document structure logically follows the actions taken to form the prototype application.

### **The Director**

Edward Verbree

### **The team coach**

Sisi Zlatanova

### **The team**

Matilde Oliveti

Godelief Abhilakh Missier

Damien Mulder

Dimitris Zervakis

Haoxiang Wu

# Contents

Preface .....	2
List of Figures .....	5
List of Tables.....	9
List of Acronyms.....	10
1. Executive Summary .....	11
1.1 System architecture .....	11
1.2. The Libelium Meshlium Xtreme Scanners.....	12
1.3 Space subdivision.....	12
1.4 Localization .....	14
1.5 Navigation .....	15
1.6 Visualization.....	16
1.7 Conclusions .....	17
2. Introduction .....	18
2.1 Introduction to the topic .....	18
2.2 Research area.....	18
2.3 Structure of the report.....	19
3. Indoor navigation applications .....	20
3.1. Introduction .....	20
3.2. Examples of an indoor navigation applications.....	20
4. Requirements.....	23
4.1 Users.....	23
4.2 Requirements tree.....	23
4.3 Constraints.....	24
4.4 Budget .....	25
4.5 Technical risk assessment.....	25
5. Conceptual design .....	27
5.1. Concept application.....	27
5.2. Concept system architecture.....	29
6. Literature research and theory .....	32
6.1. Privacy.....	32
6.2. Space subdivision.....	33
6.3. Localization .....	35
6.4. Navigation .....	48
7. The Libelium Meshlium Xtreme Scanners .....	53

7.1. Information about the Scanners.....	53
7.2. Testing with the scanners in the faculty of Architecture .....	55
7.3. Testing in ‘de Rotterdam’ .....	62
8. System Engineering.....	71
8.1. Software used .....	71
9. Space subdivision .....	72
9. 1. Implementation.....	72
10. Localization .....	83
10.1 Chosen Localization Methods .....	83
10.2 MULTILATERATION.....	83
10.3. FINGERPRINTING .....	96
11. Navigation .....	106
11.1 Implementation.....	106
12. Visualization.....	117
12.1. Application functionality.....	117
12.2. Database connection.....	118
12.3. Python localization .....	119
12.4. Route description.....	120
12.5. Unity visualization.....	120
12.6. Considerations .....	125
Conclusions .....	127
Recommendations .....	129
References .....	130
Appendix .....	133
Appendix 1. Analysis of the tests in the Faculty of Architecture.....	133
Appendix 2. Tests ‘De Rotterdam’ Results .....	137
Appendix 3. Space subdivision .....	146
Appendix 3: Network.....	152
Appendix 4. Python 2.7 code: Multilateration scripts .....	157
Appendix 5. Python 2.7 code: WIFI FINGERPRINTING SCRIPTS .....	171
Appendix 6. Code for the Android applicaton .....	175

## List of Figures

Fig. 1: Schema of Pazl (Source: Radu et al., 2013).....	21
Fig. 2: System architecture (Source: Quintas et al., 2013).....	21
Fig. 3: System architecture. (Ching et al., 2010).....	21
Fig. 4: Different display options (Source: <a href="http://www.navizon.com/its/whitepaper.pdf">http://www.navizon.com/its/whitepaper.pdf</a> )..	22
Fig. 5. Requirements tree .....	23
Fig. 6. Risk map assessment (Source: Holland & Holland Enterprises Ltd, n.d.) .....	25
Fig. 7. Example of the functions of a colleague finding application.....	27
Fig. 8. Visualization of the system concept.....	28
Fig. 9. Direct connection or connection through a middle layer.....	30
Fig. 10. Concept System Architecture.....	31
Fig.11. Indoor Spatial Models.....	33
Fig.12. Geometric representation of indoor space. (Source: OGC IndoorGML, 2014).....	34
Fig.13. Adjacency and connectivity graph. (Source: OGC IndoorGML, 2014).....	34
Fig.14. Multi-Layered Space Model (left) and an example of a Multi-Layered space representation (right). (Source: OGC IndoorGML, 2014) .....	35
Fig.15. Different empirical models for signal coverage within a building. ....	36
Fig.16. Electromagnetic intensity map making use of the Helmholtz equations. ....	37
Fig.17. Scanner placement: (a) Max Distance, (b) Concrete Square, (c) Elevator & pathways, (d) Half-building, no thick wall interference .....	38
Fig.18. Concept of area rings of signal intensity per scanner – no degradation. ....	40
Fig.19. Trilateration (2D). ....	41
Fig. 20. Trilateration with error in radii included for each scanner (S1,S2,S3). The blue area is the most probable location of the device. ....	41
Fig. 21. Multilateration.....	42
Fig.22. Triangle ABP .....	42
Fig. 23. Triangle ABP .....	43
Fig. 24. Triangulation with 3 known points .....	44
Fig. 25. Example error calculation .....	45
Fig. 26. Example of fingerprints. ....	46
Fig. 28. Square and hexagon subdivision (Afyouni et al 2012). ....	48
Fig.29. Regular grid space subdivision of ‘De Rotterdam’ building (2x2m on the left and 1x1m on the right). ....	48
Fig. 31. Example for the partitioning of building interior into rooms and its representation in dual space. ....	49
Fig.32. Example graph .....	50
Fig. 33. Adjacency matrix .....	50
Fig. 34. Adjacency list .....	50
Fig. 35. Set of labelled edges .....	50
Fig. 36. Visualization of Dijkstra and A* algorithm pathfinding on grid.....	51
Fig. 37. Example of Topographic SpaceLayer. (Source: OGC IndoorGML, 2014).....	51
Fig. 38. Indoor space mapped to IndoorGML Navigation module classes. (Source: OGC IndoorGML, 2014).....	52
Fig. 39. The Libelium Meshlium Xtreme Scanner (Source : Libelium) .....	53
Fig. 40. Conceptual schema of the working of a scanner (Source : Libelium).....	54
Fig. 41. Ways to store the gathered data (Source : Libelium).....	54

Fig. 42. Manager System (source : Libelium).....	55
Fig. 43. Testing area in red.....	55
Fig. 44. Testing environment .....	55
Fig. 45. Materials used during the tests.....	56
Fig. 46. Test set- up .....	57
Fig. 47. Scanner set- up test 1 .....	57
Fig. 48. Scanner set- up test 2 .....	57
Fig. 49. Scanner set- up test 3 .....	58
Graph 1. Result test 1 .....	59
Graph 2. Result test 1 .....	59
Graph 3. Result test 2 .....	60
Graph 4. Result test 3 .....	61
Graph 5. Result test 3 .....	61
Fig. 50. Space subdivision.....	62
Fig.51. Interior of the 16 <sup>th</sup> floor of ‘De Rotterdam’ building.....	63
Fig.52. Scanner placement test 1 .....	63
Fig.53. Set- up test 1.....	64
Graph 6. Results test 1.....	64
Graph 7. AVG RSSI values for scanner 678.....	65
Fig.54. Scanner placement test 2.....	66
Fig.55. Set-up test 2.....	66
Graph 8. Results test 2.....	67
Graph 9. The number of times a scanner has seen the three phones during test 2.....	67
Fig.56. Scanner placement test 3.....	68
Fig.57. Set-up test 3.....	68
Graph 10. Results test 3.....	69
Fig.58. Scanner placement test 4.....	69
Fig.59. Set-up test 4.....	70
Graph 11. Results test 4.....	70
Fig.60. Attribute table ‘De Rotterdam’ floor plan with semantics information.....	73
Fig. 61. Floor plan simplification.....	73
Fig. 62. Several workspaces on the 16 <sup>th</sup> floor .....	74
Fig. 63. Intuitive space subdivision.....	74
Fig. 64. Constrained Delaunay Triangulation.....	75
Fig. 65. Combination of the triangulation with the heat maps for scanner layout 1.....	76
Fig. 66. Irregular buffers according to RSSI values for scanner layout 1.....	77
Fig. 67. Space subdivision combining triangulation with heat maps for scanner layout 1.....	77
Fig. 68. Combination of the triangulation with the heat maps for scanner layout 2.....	78
Fig. 69. Irregular buffers according to RSSI values for scanner layout 2.....	78
Fig. 70. Space subdivision combining triangulation with heat maps for scanner layout 2.....	79
.....	79
Fig. 71. Range of the scanners 10 meters (left) and 4 subspaces (right).....	79
Fig. 72. Range of scanner 20 meters (left) and different subspaces (right). .....	80
Fig. 73. Automatic space subdivision with eight different subspaces.....	80
Fig. 74. Testing the automatic method for layout 2 .....	81
Fig. 75. Two different triangulations of the space .....	81
Fig. 76. Using bigger triangles, results in an even coarser subdivision .....	82
Fig. 77. Example of logs by scanner named “mesh121”.....	83
Fig. 78. Theoretical distinction between real world positions using RSSI values. ....	84

Fig. 79. Real RSSI values over time. Normally, 6 ‘distinct’ areas/plateaus should be visible, one every 5 minutes (300sec). .....	84
Fig. 80. The weights applied to RSSI values change depending on the chosen timespan – the older the values are the less credible they are. ....	85
Fig. 81. Distance $d$ from scanner and added error result in a ‘ring’ area. ....	86
Fig. 82. Relation between RSSI and distance, together with the vague error areas. ....	87
Fig. 83. Calculating the most probable localization area. ....	88
Fig. 84. Example of localization achieved. ....	89
Fig. 85. Example of combined localization. ....	90
Fig. 86. Successful localization per intersections: 0,2,3 and 4 rings intersecting. ....	93
Fig. 87. Failed localization: treatable cases. First solution (yellow) is wrong, but second suggestion (magenta) succeeds. ....	94
Fig. 88. Failed localization: non-treated cases. Both first and second solutions fail. ....	94
Fig. 89. Successful localizations for different scanner layouts. ....	95
Fig. 90. Fingerprints: (a) layout 1; (b) layout 2. ....	97
Fig. 91. Sampling points in different grids: (a) 4 by 4 ; (b) 2 by 2 .....	98
Fig. 92. Heatmaps: (a) layout 1; (b) layout 2 .....	100
Fig. 93. Examples of localization result of layout 1 with 2 by 2 grid in subdivision (a) Successful localization; (b) Failed localization. ....	103
Fig. 94. Functional layer (left) and Navigational layer (right).....	107
Fig. 95. Range of the scanner of 20 meters for layout 1 and 2.....	107
Fig. 96. Implementation of the manual derivation of the network on the intuitive subdivision.....	108
Fig. 97. Subdivision based on the range of the scanners, after removing the functional layer. ....	108
Fig. 98. Concept for the semi- automatic derivation of the network.....	109
Fig. 99. Implementation of the semi- automatic method on the subdivision based on the range .....	109
Fig. 100. Implementation of the semi- automatic method on the subdivision based on the range .....	110
Fig. 101. The final network .....	110
Fig. 102. Implementation of the automatic method on the subdivision based on the range .....	111
.....	
Fig. 103. The final network .....	111
Fig. 104. Implementation of an automatic method on the triangulated floorspace.....	112
Fig. 105. The network used for the application.....	112
Fig. 106. Generated network with nodes.....	113
Fig. 107. Example adjacency list.....	113
Fig. 108. Possible route description statements based on present landmarks .....	115
Fig. 109. Photos ‘de Rotterdam’, arrows indicate route description statements “left, right, right” .....	115
Fig. 110. Navigation based on cardinal direction.....	116
Fig. 111. Main screen, department selection and colleague selection in mobile application .....	117
.....	
Fig. 112. Selection overview and route description screen in mobile application.....	118
Fig. 113. 2D, 2.5D and 3D visualisation of routing within ‘de Rotterdam’.....	121
Fig. 114. Overview of the unity scene.....	122
Fig. 115. Table of node positions on floor 14 .....	124
Fig. 116. Set- up Test 1 .....	133

Fig. 117. Set- up test.....	134
Fig. 118. Set- up test 3.....	134



## List of Tables

Table 1. Point position accuracy resulted from different error of measured distance.....	46
Table 2. Pros and cons of localization methodologies.....	48
Table 3. Pros and cons of grid and network space subdivision models.....	50
Table 4: Components and tools used to create the prototype .....	72
Table 5. Pros and cons of the intuitive and automatic space subdivision.....	83
Table 6. Localization success rate (%) on same room as ground-truth point. If the point lies in the same room as the room returned by the algorithm, it is a success hit.....	92
Table 7. Localization success rate (%) on same room OR neighboring rooms. If the point lies in the same room as the room returned by the algorithm OR any of its adjacent rooms, it is a success hit.....	93
Table 8. Localization success rate (%) on same room as ground-truth point. (automatic subdivision, only layout 1 tested).....	93
Table 9. Localization success rate (%) through combined subdivisions. (intuitive and automatic, only layout 1 tested).....	93
Table 10. Heat maps testing results.....	102
Table 11. Localization result in space subdivision.....	103
Table 12. Improved localization result in space subdivision.....	105
Table 13. Changes of successful rate for different set-ups.....	106
Table 14. Departments table in MySQL database.....	119
Table 15. Phones table in MySQL database.....	119
Table 16. Route description based on node-pairs.....	121
Table 17. Route description based on node-triples.....	121
Table 18. Implementations of the components in the prototype.....	128

## List of Acronyms

AoA	Angle of Arrival
AP	Access Point
CAD	Computer-Aided Drafting
DBMS	Database Management System
FDTD	Finite Difference Time Domain
GIS	Geographic Information System
GML	Geographic Markup Language
GPS	Global Positioning System
IPS	Indoor Positioning Systems
ISO	International Organization for Standardization
LBS	Location Based Service
MAC	Media Access Control
MLSM	Multi-Layered Space Model
NRG	Node Relation Graph
OGC	Open Geospatial Consortium
OS	Operating System
PHP	Hypertext Preprocessor
RSSI	Received Signal Strength Indicator
SQL	Structured Query Language
SPKF	Sigma-point Kalman filters

# 1. Executive Summary

The project 'De Rotterdam' aims to provide the Municipality of Rotterdam with a solution to the problem its employees face, when needing to contact and meet fellow team members in the vast new environment of 'De Rotterdam' building. In fact, in this building, employees do not all have a fixed workplace to work at, but can choose to work at flexible workplaces. This makes it hard for employees to find their colleagues, especially since the building has 44 floors. In addition, another challenge is addressed, which stems from the unawareness of employees about the availability of free workspaces in 'De Rotterdam', which will cost them time and can cause frustration.

The team comprised for this project has been asked to develop a smartphone application with an easy to use interface that can locate its user, as well as the colleague the user wants to find, in a reasonable time frame with the help of Wi-Fi monitoring. Additionally, dependable navigation should be provided with a route description. As an agreed limitation, given that most of the employees use Samsung smartphones, the application will be aimed for Android software devices.

The requirements in this project are decided by three parties: the client, the coaches and the team itself. For the client, a working prototype is expected that fulfils the functions described above. For the coaches, it is important that the students work as a team, where every individual has a distinct technical role. For the team, the foremost purpose is to create an end-product that they can be proud of. To achieve this, everyone in the team has to be active and creative. Besides the result, a relaxed and non-stressful way of working is desired, while the educational learning factor remains high.

The scientific research for this project focuses on the extent to which it is possible to localize a detected device through Wi-Fi monitoring. For this purpose Wi-Fi scanners are used and data needs to be consistently collected. A number of localization methods can be applied and the final result is displayed to the client's mobile device.

## 1.1 SYSTEM ARCHITECTURE

In our design, its major components include the programming language, the operating system (OS), a Database Management System (DBMS) and a Web Service. The OS for this smartphone application will be Android, as requested by the Municipality of Rotterdam. Next, the programming language will be Android Java with JavaScript and PHP connections and Python, which the group members are more familiar with. For this application that requires a lot of data, a DBMS is also necessary, to keep the data organized in one place and to be able to query fast. Although the database can be stored on either an external server or locally stored on the smartphones, an external server will be a better choice in this case, considering the application's performance on the smartphones, since the data storage requires a lot of memory. Besides, in order to further make the app lighter on the smartphones, a web service, acting as a middle layer between the client and the database, can also be considered, by which all the calculations including localization and path-finding algorithms on the database can be called from the external server.

For desirable functions, foremost the application should localize an employee. This localizing is done using Wi-Fi monitoring. A localization algorithm determines the localization of the user and the employee that he/she wants to be found. When the employee is found, the user wants to be navigated to the colleague. A shortest path algorithm takes the

positions of the user and the target employee and calculates the shortest route to take. This route will be visualized by a rendering program and sent to the smartphone.

## 1.2. THE LIBELIUM MESHLIUM XTREME SCANNERS

The hardware used in this project consists of 4 Libelium Meshlium Xtreme Scanners, which are used to scan for Wi-Fi-probes of the smartphones. The data received by the scanners always contains the MAC address of the scanned device, which allows to identify it uniquely, the strength of the signal (RSSI), which gives the average distance of the device from the scanning point, the vendor of the smartphone (Apple, Nokia, etc.) and the TimeStamp, which indicates the date and time the data was collected.

The collected data can be either stored locally on the Meshlium scanner or stored in an external database.

Three tests with the 4 Libelium Meshlium Xtreme Scanners were performed in the faculty of Architecture in order to find out the range of scanning of a Wi-Fi monitor, the influence of obstacles on the signal strength and in order to distinguish areas based on signal strengths. This environment was chosen because the space is similar to the environment of 'De Rotterdam'. In order to keep all tests consistent, a few things were taken into consideration: 4 Meshlium Xtreme and 3 Samsung Smartphones were utilized for the all four tests, the scanners and the phones were time synchronized, the scanners were all set on a scanning time interval of 30 seconds and the data was collected in each point for a time interval of 5 minutes.

Then six tests were performed in 'De Rotterdam' building were carried out to collect the data for the implementation of the so-called 'Catch-a-colleague' application. The tests were held in the 16<sup>th</sup> floor of the building, where the environment consists mainly of open spaces with free workspaces and small rooms made of glass and thin walls.

Four different scanner layouts were tested. In the first scanner layout the scanners are placed at the four corners of the building. In the second scanner layout the scanners are placed close to the corners of the central empty-space rectangle which includes the elevator area, stairs and non-working areas. In the third layout the scanners are placed around the concrete block near the elevator and pathways, which are better covered in this case. In the fourth scanner layout, the four scanners are placed only in half of the floorplan to explore the impact of having a denser scanner population per floor.

The application developed in this project consists of several components: space subdivision, localization, navigation and visualization. All of them will be addressed in detail in the next paragraphs.

## 1.3 SPACE SUBDIVISION

An important component of the 'Catch-a-Colleague' application is the space subdivision of indoor space, which is fundamental for correctly guiding an employee to the location of another colleague but also for testing the localization algorithm. Since the indoor environment is much more complex than the outdoor environment, different aspects have to be taken into account while modelling and subdividing indoor space, such as obstacles like furniture, columns and walls. Different spatial models can be chosen to model indoor environment: geometric models for representing the shape and the metric properties of spatial objects,

topological models to highlight the relation between spatial objects, whereas semantic models to focus on the meaning of spatial features.

First of all, for subdividing the space in 'De Rotterdam' building a few requirements are set-up by the team: the subdivisions should all have the same size, should consist of around eight subdivisions (based on the localization accuracy), should not distinguish between rooms/ open space and should not be too detailed. In the end, two different implementations are carried out by the team: the intuitive space subdivision and the automatic space subdivision, which is based on the Multi- Layered Space Model from IndoorGML.

The intuitive space subdivision is based on subdividing the space in a human-understandable way, considering the characteristics of the building (obstacles, rooms, etc.), visibility criteria (e.g. line of sight) and the usage of space (workspaces). Despite this solution might lead to a better human-understandable result, it is quite hard to be implemented in an automatic way, since the environment must be modelled accurately. Furthermore, it can be time consuming, because it requires manual editing of the subdivisions. The subdivision is made in CAD software and then the different regions are converted into Shapefile format using GIS software. In the end eight different subspaces are created, based on the different workspaces that can be distinguished in the floorplan.

The automatic space subdivision is carried out performing a triangulation of the indoor space, since the team has experience with this method. The Constrained Delaunay Triangulation has been chosen to decompose the polygon of the floor plan into non-overlapping triangles. The Triangle software package with Pyshape and Shapely Python libraries are used to script the algorithm. Later, the triangles created by the triangulation are then combined with the range of the wifi scanners, in order to create a space subdivision that fit well with the accuracy of the localization method. Buffers with a radius of 10 and 20 meters, taken from the heatmaps of the fingerprinting localization method, are built around each scanner. All the triangles that fall into the circle were considered being part of one subspace. In the end all the triangles belonging to each subspace are merged together and the overlapping polygons are manually deleted, so that in the end each subspace have about the same size. The automatic method can thus better be referred to as semi-automatic. With this method eight different subspaces were created in the end.

The triangulation was also combined with the heat maps, but with this approach just four or five subspaces are created. Since it leads to a rather too coarse subdivision, which is not suitable for the localization, this approach has not been tested and automatically implemented in code.

Both the space subdivision has been tested in the localization algorithm. A slightly better result in the localization has been registered with the semi- automatic space subdivision because it takes into account the range of the scanners, which are not considered by the intuitive one. However, the intuitive space subdivision seems in the end to be the most suitable for being used in the navigation, because it is based on the usage of space. By dividing the space into different workspace, is an more understandable way for a human to navigate in an office space.

## 1.4 LOCALIZATION

Among the functions, the localization algorithm is the core and also the most difficult part in the whole project. Thus, an extended research on localization techniques was performed, based on literature and case studies. In the end, two localization methods are explored and implemented by the team: the multilateration method and Wi-Fi fingerprinting method.

In the multilateration approach, a multi-metric function that mainly takes into account RSSI values and roughly translates them into distances of the devices from scanners is constructed. It is the process of determining a relative unknown position of the device at question, using the geometry of spheres or circles, whose radius is described by the above function. At first the trustability of the RSSI values perceived by the scanners needs to be assessed. For this reason, a logarithmic function is derived. The function applies a weight to each RSSI value, based on how recent it is to the time of request. After the weights are calculated for each RSSI value available, the weighted average of them is computed that yields one RSSI value for each scanner. The next step consists in translating those RSSI values into distance, even if this translation is not easy to handle and in many cases it has been proven that exact positioning through this method is impossible. Nevertheless, a function was constructed to make the multilateration method feasible to an area/room extent. Normally, in this method, at least 3 circles need to intersect in order to achieve localization. In indoor environments with a small number of scanners this level of availability is quite difficult to achieve. In order to overcome this, an indicator is applied on each ring that defines its priority in choosing it as a best option, even when tri-/multi-lateration is not achieved. The outcome of the algorithm is an area that localizes the device in question within it. The subdivision that matches best the resulting area is the one returned as the final solution of localization. As an added feature, it is possible that after the algorithm returns the first solution, it can also return a second option of an adjacent room that possibly the device lies.

The fingerprinting approach consists of mainly two phases: the training phase and the matching phase. In the training phase, the given area is divided into many small cells, and for each of them an RSSI value is rather directly measured or computed by interpolation. Then heatmaps with these values are created. In the matching phase, live RSSI values can be compared, using a matching algorithm to find the best match with the training database (fingerprints), created in the previous phase.

For the implementation of this method, before the raw data collected in 'De Rotterdam' building can be used, some processing is carried out and a local coordinate system is defined using two different grids (2x2m and 4x4m). Since not for every cell of the grid an RSSI value has been measured, interpolation is performed, using the Scipy Python library. Not all the sampling points are used in the interpolation, since some of them (testing points) are used to test the accuracy of the heatmaps. Then, the Nearest Neighbor matching method is used to match between the recorded fingerprints (sampling points) in the heat maps and the testing live fingerprints (testing points). Location with the least sum of squared differences is assumed to be the best match. As for the multilateration method, the localization can be improved when given a second choice or even a third choice.

The multilateration method relies heavily on the chosen function which models the translation of RSSI values into distance. The function itself is dependent on the scanner placement and mostly on the surrounding physical environment. Furthermore, the solution can have varying success rate depending on the space subdivision and/or combined solution. Therefore, indoors localization through this process is a multi-layered problem which takes into account a

number of factors that are difficult to model, but can achieve a good outcome with a high level of automation and environment modeling.

On the other hand, fingerprinting method is a precise method and close to the real distribution of values. It can be updated by user input. However, it is quite time-consuming to collect the data and it strictly depends on the scanner layout.

## 1.5 NAVIGATION

In order to enable the user to find their colleague, navigation is needed to communicate the route to the user. For accomplishing this task, a description of how the subdivided spaces are connected is needed. After having analyzed the main approaches for generating the navigation system, the network approach was selected for 'De Rotterdam' building since it fits better with the characteristics of the building, it is easy to design (few nodes and edges needed) and not high localization accuracy is needed.

In total, three different networks have been implemented by the team: the manual network, the semi-automatic network and the automatic network.

Since the manual network is manually drawn using a CAD software, it can be easily adapted to the characteristics of the building (obstacles, rooms, etc.).

The semi-automatic method considers the building geometry. Certain nodes are necessary for an efficient routing for any floorplan in the MidTower of 'De Rotterdam' building, regardless of the space subdivision. For this reason, a 'basic routing' is created as a part of the network. It consists of nodes in and around the buildings core, enabling navigation from and to the elevators and staircases. Additional nodes for the routing around the core of the building are added to enable an effective routing. After this basic routing is in place, the subspaces resulting from the space subdivision can be taken into account. For each subspace the centroid is computed using the Python library Shapely. The script thus then searches for the center points of the subdivisions and connect these points with the route around the core.

The automatic network does not consider the building geometry. It uses the center points of the subdivisions as a starting point. After computing the centroid, each centroid is connected with the centroids of the neighbouring subspaces and in this way the network has been generated. In many cases the network crosses holes which represent the rooms left out the subdivision, since they are seen as obstacles, where people cannot walk through.

Once the network is generated, a route computation is needed from the user's position towards the target position. When both positions have been appointed to a certain node, a computation of the shortest path between these nodes can be performed. The user's position will then be defined as a start node and the target's position will be defined as the end node. A path finding algorithm will search for adjacent nodes around the start node, until the end node has been found.

Different path finding algorithms exist, varying on complexity and computational efficiency, each more suitable for a certain application. Since a starting point is already known, single source shortest paths algorithms are considered for a weighted and undirected graph. In our case the graph is undirected because any movement through the network can be done in both directions. In our implementation, the Dijkstra path finding algorithm has been used, since it is easy to implement and the team has experience it. A prewritten Dijkstra algorithm scripted in Python has been used, which takes as input an adjacency list with the nodes and the edge lengths. The Dijkstra algorithm searches through adjacent nodes, based on a priority queue which extracts the nodes with a minimum weight (distance) from the source. This means that equal amounts of nodes are visited in all directions, until the end node has been found.

## 1.6 VISUALIZATION

In order to navigate an employee towards a colleague, a visualisation might be the most direct way of achieving this. The route the colleague has to follow will be displayed in the application, in which all the loose components are combined. The application is developed for Android, as the employees of the Rotterdam municipality are granted mobile phones which run on that platform. Since, there are around 3000 employees working for the municipality of Rotterdam, all of whom can be present in the office, it should be easy to select the colleague one is looking for.

The first step is to acquire the mac-addresses of the searching and the searched colleague, which are stored in a database. In this case MySQL workbench is used and two tables are created: one table storing data about departments and another table for individual phones. To select the searched colleague, the relevant department can be chosen out of a list of departments. Then following step is to select a colleague that belongs to the selected department. Then, a connection to the database has to be made in order to retrieve the mac-address of the searched colleague. Frequently updating and validating the database with the correct mac-addresses for each colleague is crucial for the functionality of the system. Since the users' location is also required, its mac-address should also be retrieve. Once these values are found, the localization script can be requested to run with the two mac-addresses as input nodes to retrieve the shortest route. In order to query these tables from the application, PHP is used. The PHP-file is stored on an external server.

Similarly, another PHP file is called, in order to call the python script containing the localization algorithm on a remote server, to run with two mac-adresses as input. Once a location is returned, the Dijkstra's shortest-path algorithm is run on these two nodes and in the end a route given as a sequence of node numbers is returned.

When the route is known, a visualisation which can aid the user with its navigation can be created. To do so, the route should be rendered and accompanied by an illustration of relevant building parts.

The visualisation can be done rather in 2D, 2.5D or 3D, depending on the demands and the complexity of the building geometry. Since a 3D visualisation can show more information about the building geometry, which may help the user, it is chosen to be implemented in the project. With a 3D visualization, (parts of) a 3D-model can be depicted on the mobile phone, while nodes and lines can be rendered on top of this. Showing or switching between different floor levels (visible buildings parts) and levels of detail (layers) belongs to the possibilities. However, it could be heavy to run as mobile application.

In order to render the geometry and routing the team has decided to use the 3D engine Unity3D. Unity3D is known to be capable of visualizing the geometry and routing and, more importantly, it is compatible with Android applications. Within Unity3D a scene is created, consisting of a 3Dmodel, lighting and textures. On top of this the route will be renderer. Since the model created for the three floor plans is not very rich in semantics nor is it geometrically detailed, the geometry is directly stored on the phone rather than on the server. To visualize the route clearly, only the floor plan, on which the searched colleague is present, is used.

The camera is used to determine which part of the geometry is visible and in what way. The team has decided to use a bird's eye view perspective as it created the most oversight of the building. Although the entire floor plan is visible from the initial view, users may wish to change the camera position in order to get more information of the building geometry.



## 1.7 CONCLUSIONS

To conclude, the project aims to help employees in 'De Rotterdam' building to find their colleagues and free workspaces. The objective is to develop a working application prototype for Android smartphones with an easy to use interface in 3D-view that can locate its user and colleagues and provide a dependable navigation with route description with the help of Wi-Fi monitoring. There are requirements from the client, the coaches and the team, that have to be fulfilled in order for this project to be a success. Foremost, the team has tried to achieve a working prototype for the application Catch-a-Colleague that complies with all the requirements mentioned above.

In the development of the application, privacy issues may arise since some issues related to the use of the user's location information may occur. However, smartphones have been provided to the employees by the Municipality of Rotterdam, thus many privacy issues may be avoided in this case. The employee's permission should still be asked in advance, as well as the purpose of collecting the MAC address of the devices should be clearly stated and personal data should be sufficiently protected from unauthorized use.

Making a conclusive remark for each of the loose components, it can be said that the multilateration method relies on the scanner placement and on the scanner layout, which may affect each other. RSSI has been discovered to be just an indicator which is not directly translatable into distance. In the end with multilateration it is possible to achieve a good outcome if the environment is modelled.

For the fingerprinting localization method the placement of the scanners define the result of the heatmaps and therefore it is a deciding factor for this approach.

Regarding the space subdivision, the intuitive space subdivision seems in the end to be the most suitable for being used in the navigation, because it is based on the usage of space and it is more human-understandable.

For the navigation component, by comparing all the different networks with each other, the manual network seems to be the most effective since it can be easily adapted to the characteristics of the building (obstacles, rooms, etc.). Further investigation needs to be performed in order to achieve a network that can represent a manual one in terms of efficiency for the scope of the application.

Finally, for the visualization component, a 3D model on the mobile phone may help the user as it shows additional information about the building geometry. For indoor environments with complex spaces, route descriptions may be of benefit to the user.

In general it can be concluded that the challenges of creating an indoor navigation application do not only lie in the loose components. The integration of the system requires both some technical knowledge and resources. Considering the time frame allowed for the project and the research scope, a satisfactory result is achieved.

## 2. Introduction

### 2.1 INTRODUCTION TO THE TOPIC

The Municipality of Rotterdam is moving to their new workspace: the new office building 'De Rotterdam' in Rotterdam. This building consists of three towers, of which the municipality will be using the so called MidTower, partially. An important part of moving to the new offices involves making use of flexible workspaces as approximately 3200 colleagues will be working on 2500 workspaces. This means that employees will not have their own workspace, but will work on any available workspace on a given day. A consequence of this is that finding your colleagues within the building is a difficult issue, as they may be working in different areas on different floors of the building. Employees are supplied with mobile phones by the Municipality, which could be used in a solution which aids the user to find his/her colleague. Therefore, the main objective of this project is to develop a mobile application which enables the user to find a colleague within 'De Rotterdam' building.

### 2.2 RESEARCH AREA

#### 2.2.1. Objective

The main objective of the project is to:

*“Create an Android smartphone application for the Municipality of Rotterdam to locate their colleagues in 'De Rotterdam' building with the help of Wi-Fi monitoring”.*

#### 2.2.2 Research area

##### *Research question*

The research question corresponding to the main objective is:

*“To which extent is it possible to localize a colleague using Wi-Fi Monitoring on a single floor level in 'De Rotterdam' building?”*

The research of the team will be focused on the localization part of the application, since a precise localization is required for the goal of the application.

In order to give an answer to this question, research and testing needs to be done. Since the endproduct is one application, where all the components such as navigation, space subdivision and visualization are integrated, the research question can only be answered by having also subquestions/ research areas concerning these other components.

These subquestions/ research areas are:

1. An investigation of the positional accuracy of Wi-Fi monitoring techniques.
2. How can you model the relationship between signal strength (RSSI) and distance?
3. What is the best space-subdivision method for 'De Rotterdam' building?
4. How can you visualize the route to take?
5. How to integrate the different components (localization, navigation and visualization?)

#### 2.2.3. Research approach

In order to create the end result, a working prototype, several steps are done by the team. After stating the research question, literature research was done. That included reading scientific papers, as well as meetings with the project coaches and client. During the research, several tests were performed with the hardware provided, in different testing environments. Concepts were thought of and implemented. Results were gathered to test the implementation and to be able to answer the research question. Furthermore external help was sought in the

form of a meeting with the project manager of the Libelium Company about the Meshlium scanners and a (possible) meeting with the E-Semble company for helping with the application.

### 2.3 STRUCTURE OF THE REPORT

This report is structured as follows. First, an introduction to indoor navigation applications is given. Then the guidelines for this project are provided, such as the requirements, the users, the constraints and the budget. Then the concept will be explained into more detail. After that, a lot of literature research done by the team is summarized. In the next chapter more can be read about the scanners and the performed test. The chapter that follows explains in a technical way, the implementation of the components to make the application and the results of the process are displayed. The last part of the report contains the conclusion with the answer to the research question, as well as recommendations for further research.

## 3. Indoor navigation applications

### 3.1. INTRODUCTION

The 'Catch-a-Colleague' application which is developed for this project falls into the category of Location-Based Services (LBS), which can be defined as services that integrate a mobile device's location or position with other information so as to provide added value to a user (Schiller et al., 2004). LBS are extensively used in outdoor applications. But estimating the location of people and tracking them in an indoor environment is still a challenge, since the accuracy of explicit positioning sensors such as GPS is often limited for indoor environments (Paul and Wan, 2008).

Nowadays, the ability to navigate people in indoor environments has become increasingly important for a large number of applications, since the average person spends approximately 90% of his/her time inside buildings, indoor environments play a particularly central role in human activities (Jenkins et al. 1992).

Several different approaches for indoor localization exist, using a variety of technologies such as ultrasonic sound, UWB radio, Wi-Fi, RFID, Bluetooth, Infrared, etc. (Quintas et al., 2013). In particular, Wi-Fi is an attractive positioning technology due to the widely deployed Wi-Fi access points (APs) and the growing number of Wi-Fi-enabled mobile devices on the market. Wi-Fi APs can be found almost in every public building, such as in offices, hotels and shopping centres, etc. (Ching et al., 2010).

Using smartphones for accurate indoor localization opens a new frontier of mobile services, offering enormous opportunities to enhance users' experiences in indoor environments. Despite significant efforts on indoor localization from universities and industries in the past two decades, highly accurate and practical smartphone-based indoor localization remains an open problem (Liu et al., 2013).

From literature review emerged that so far, not so many indoor navigation system based on Wi-Fi technology have been developed, especially in office environments. Some commercial companies (e.g. Insoft) provides solutions for indoor office environment positioning and navigation but most of them combines different sensors (e.g. accelerometer, gyro, camera, Wi-Fi, bluetooth, etc.) in order to achieve better accuracy. In addition, the number of system for indoor navigation using robots in office environments have also been recently implemented (Marder-Epstein et al., 2010; Biswas and Veloso, 2010), but an overall solution based on a single technology has not yet been determined.

Considering the localization component, most of the case studies found in the literature review apply Wi-Fi fingerprinting technique since it can reach a relatively high accuracy in the order of magnitude of meters, whereas Wi-Fi monitoring is not much implemented, due to the accuracy.

### 3.2. EXAMPLES OF AN INDOOR NAVIGATION APPLICATIONS

Radu et al. (Radu et al., 2013) developed Pazl, a mobile crowdsensing-based indoor Wi-Fi monitoring system that is enabled by a hybrid localization mechanism, which integrates the best aspects of pedestrian dead reckoning and WiFi fingerprinting (see Figure 1). Their focus is on indoor environments with multitude of access points (APs), such as shopping malls and hospitals. Pazl relies on crowdsourcing for constructing the Wi-Fi fingerprint database and it has been implemented through a combination of an Android mobile application and cloud backend application on Google App Engine.

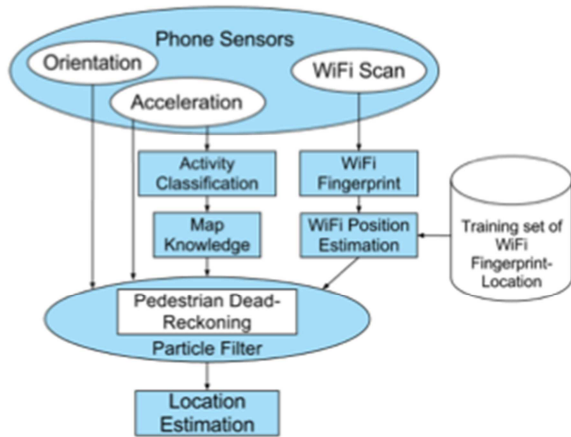


Fig. 1: Schema of Pazl  
(Source: Radu et al., 2013).



Fig. 2: System architecture  
(Source: Quintas et al., 2013).

Another indoor application for mobile phones was developed by Quintas et al. (Quintas et al., 2013) in 2013, using Android OS, localization techniques and server side logic to do the localization inside buildings. A general overview of the system's architecture can be seen in Fig. 2.

At the University of New South Wales in 2010 developments were made on an application to provide students with location based services, such as finding a lecture room, finding for example the nearest vending machine using Wi-Fi positioning technology (see Figure 3).

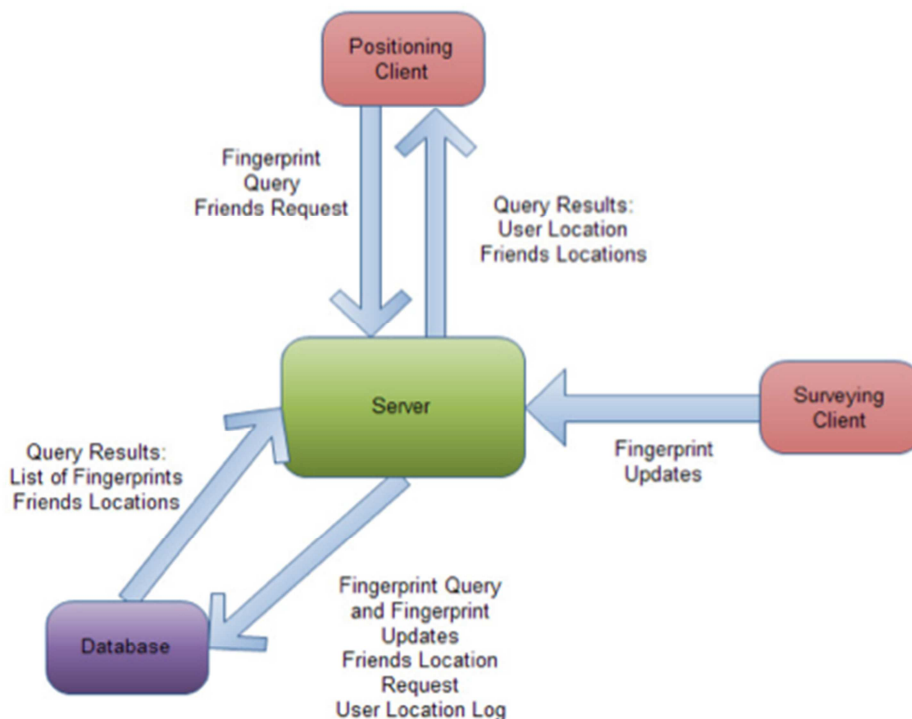


Fig. 3: System architecture. (Ching et al., 2010)

Concerning the visualization component, most of the navigation programs use primarily 2D plans for displaying the route, but in recent year some 3D interfaces have been developed for several applications e.g. for emergency response, hospitals, shopping malls, airports, etc. (Meijers, Zlatanova & Preifer 2005). For instance, an app developed by Navizon provides different display options: Buddy Radar which gives users a real-time display of where the friendly devices are located in reference to the phone's position. An overhead view is displayed, showing the location of devices on a 2D floor plan. Or Google Map's View, which shows device locations overlaid on a map of the area (see Figure 4).



*Fig. 4: Different display options (Source: <http://www.navizon.com/its/whitepaper.pdf>).*

For the current project the focus lies on creating an Android application for localizing by Wi-Fi monitoring and visualization of the navigation in 3D. The next chapters will provide more details about the concept.

## 4. Requirements

This chapter describes the guidelines that were taken into account for the application into more detail. These guidelines helped structure the project.

### 4.1 USERS

An important aspect for creating an application is to define who its users are. In the current project users are the employees of the Municipality of Rotterdam the main users. They will work in the MidTower of 'De Rotterdam' building. Phones are provided by the Municipality, so they do not use their own phones. Basically, they want to be able to find a colleague during their working day, through an application on their mobile phone.

Another user type could be visitors who have a meeting with an employee. They might probably not know the building and for them it will be even harder to find someone in the MidTower. If they install the application at arrival in 'De Rotterdam', it can help them guide them through to the building to the right floor for their meeting.

### 4.2 REQUIREMENTS TREE

The requirements of this project, as previously mentioned in the project plan, are decided by three parties: the team, the coaches and the client. A tree version of them is shown in Figure 5.

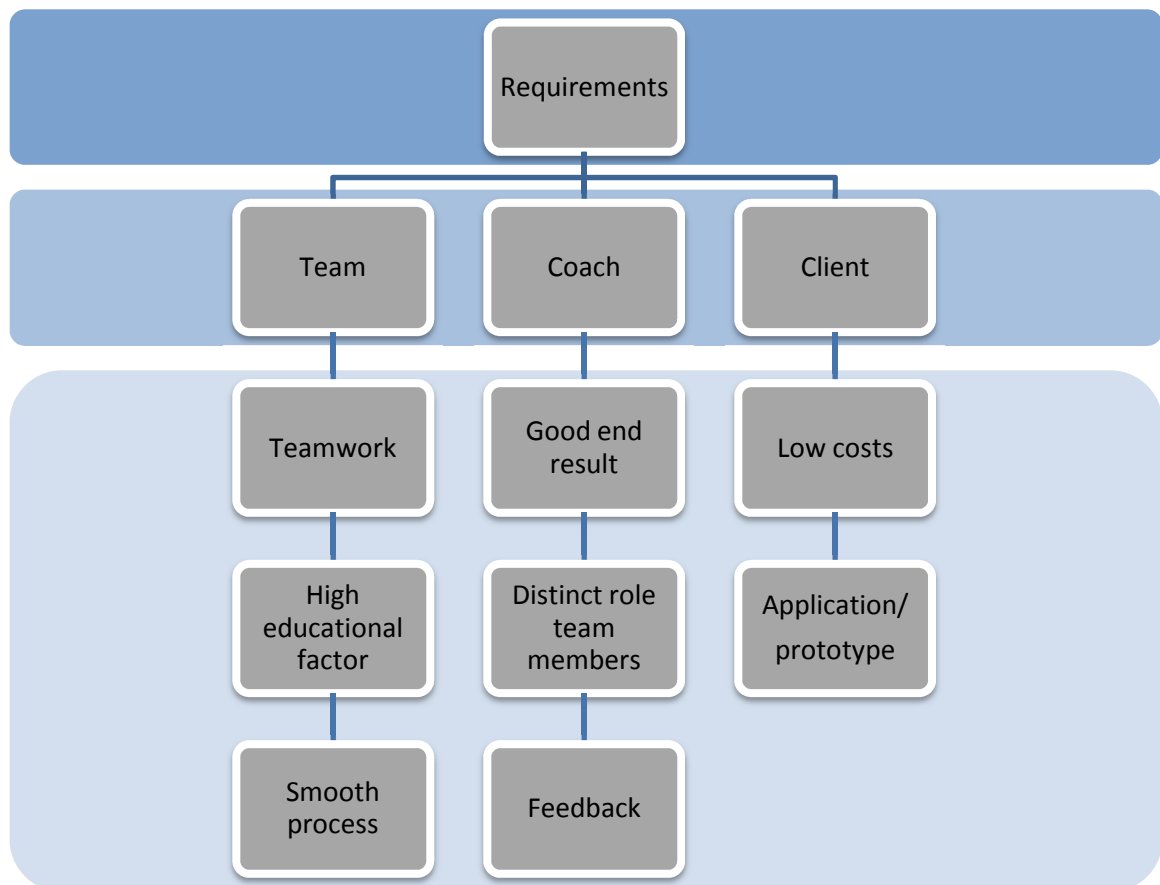


Fig.5. Requirements tree

For the **team**, value is placed on the final outcome and grade, that much can be learned as well as to have a good team spirit and acquire a lot of new knowledge during this project. In fact, the main requirements for the team concern creating an end-product to be proud of, working in a relaxed and non-stressful way, remaining creative and learning a lot during the project.

The **coaches** mostly want the team to work in a team and where every individual has their distinct technical role. In addition, they also require that the team keeps them updated of the achievements, delivers a good end-product and applies the knowledge acquired during the 1<sup>st</sup> year.

The **client**, the Municipality of Rotterdam in this case, wants to have a light application to locate its colleagues, with a clear and an easy-to-use interface. According to the client, the application should run on Android devices.

#### 4.3 CONSTRAINTS

Constraints are unavoidable factors that must be taken into consideration by the team and the client to form a circle of agreement of what can be achieved. Some of them may be unchangeable, whilst others might be managed so that they can be erased or replaced for the team to be able to explore different paths.

The constraints perceived by the team are presented below grouped by user acceptability and technology used.

##### User acceptability:

- The employees might leave their smartphones on the desk while they are away (resulting in wrong assumptions by the application)
- Not every employee will turn on the Wi- Fi which will make them impossible to be found

##### Technology used:

- The number of the scanners provided for this project might not be enough to cover the whole building/ floor
- The structure of the building or the layout of the floors might somehow affect the accuracy of the Wi-Fi scanners (in some areas the Wi- Fi signal might be blocked out)
- The accessibility of the building for the team to test the prototype
- The hardware (scanners/ smartphones) might be of poor condition or their accuracy and thus not good enough to perform a perfect positioning
- The possible layout of the Wi-Fi access points might not guarantee a satisfying result

##### Assumptions

Considering the constraints, some **assumptions** have been made by the team:

- For the Wi-Fi Monitoring only 4 Meshlium scanners are utilized
- Every employee has a Samsung smartphone, provided by the Municipality. The app will thus be coded in Android.
- Every employee carries their smartphone with them in active mode.
- Tests will only be carried out on one or max three floors, namely the 14<sup>th</sup>, 15<sup>th</sup> and the 16<sup>th</sup> floor.



#### 4.4 BUDGET

The resources for the project can be summed up in the following groups:

- *Hardware Resources:* 4 Libelium Meshlium Xtreme scanners and three Samsung Smartphones provided by the TU Delft University.
- *Human Resources:* the team (5 MSc students with different backgrounds), the coaches and the client (the Municipality of Rotterdam).
- *Software Resources:* the Android application, database (MySQL, Postgres), server (provided by Wilko Quack), Unity 3D game engine.

Apart from the hire of 4 Meshlium scanners from the company Libelium, no extra budget is expected for the project.

The performance predicted by the team is mainly related to being able to locate a person with the hardware provided, with a certain accuracy to make the ‘Catch-a-Colleague’ application working. The result to achieve is a working prototype.

#### 4.5 TECHNICAL RISK ASSESSMENT



*Fig. 6. Risk map assessment  
(Source: Holland & Holland Enterprises Ltd, n.d.)*

A risk map assessment is basically an iterative cyclical process containing the objectives and the process as planned at the start (see Figure 6). The further the project develops, the more risks can arise. It is important to think about the potential risks in an early stage, to prevent them or to take actions against them. The monitoring and keeping control are part of the quality management and are connected to the former stated objectives.

Risks in this project plan are/could be:

- to not have a 3D- model in time.

For the 3D-model the team is dependent on the client. If the client is not able to deliver the model in time, the visualization will be in 2D or in a simplified 3D- model, made by the team itself. The application could be less clear in 2D. A solution would be to start early with making a 3D-model by ourselves and to keep contact with the client about the status of the 3D-model.

- to not have access to the 'De Rotterdam' building.

If the team is not able to test in 'De Rotterdam' building, because the access to the building is difficult because of the fact that the building is partially under construction, the team has to find another location similar to this high-rise building. An example of that could be the Faculty of Electrical Engineering, Mathematics and Computer Science or the testing can be done in the old building of the Municipality of Rotterdam.

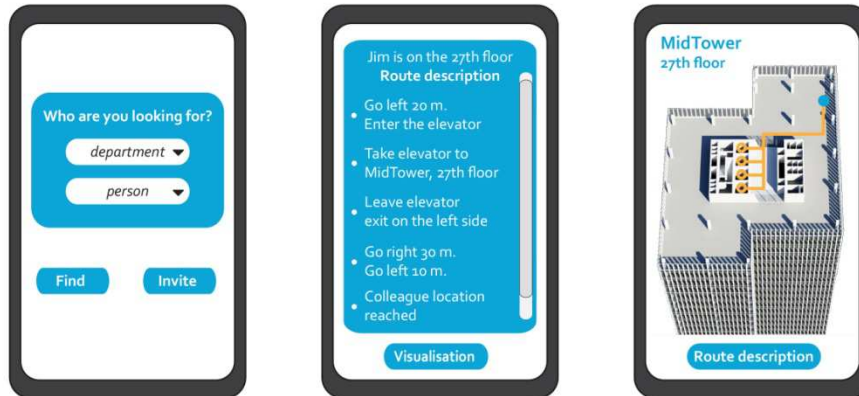
- to not be able to develop an accurate enough localization algorithm.

If the localization algorithm is not accurate enough, the positioning would not be right because the positions can be far off. Then the application would not work properly. Under the circumstances (the assigned time for this project, the knowledge of the team) the team tries to develop a localization algorithm as best as the team can.

## 5. Conceptual design

### 5.1. CONCEPT APPLICATION

An example of how this kind of application could work is illustrated in Figure 7 below.



*Fig. 7. Example of the functions of a colleague finding application*

To achieve this functionality, a system will be developed which consists of the different components needed to localize and navigate employees throughout the building. The system can be divided in three different components.

Firstly, the Localization component is introduced, which determines the location of a mobile phone. In the project Meshlium Wi- Fi scanners will be used to scan for Wi- Fi-probes of the smartphones.

Secondly, a Navigation component will determine how a user can reach the position of its target. To do so, a floor plan subdivision, a network generation and a route calculation have to be performed.

Finally, a Visualisation component will illustrate the building and the calculated route on the mobile device, in order to guide the user to its destination.

These components can be further divided in steps that need to be taken during the process, in order to develop the working application. In the listing below, an overview is given of how these steps together can form the system.

The steps in the system concept are as following (see Figure 8):

1. Multiple Meshlium scanners are continuously monitoring mobile phones
2. An external database connection enables the combining of scanner data
3. A user searches a target, the mobile application requests a position
4. A localization algorithm is run on the data to find the targets phone
5. Localization algorithm appoints the target to a subdivided part of the building
6. Localization algorithm appoints the user to a subdivided part of the building
7. A shortest path algorithm is run on the external server
8. The resulting route is the input for the 3D engine
9. The relevant building geometry is loaded in the 3D engine
10. 3D engine renders route and geometry and sends the scene data to the smartphone

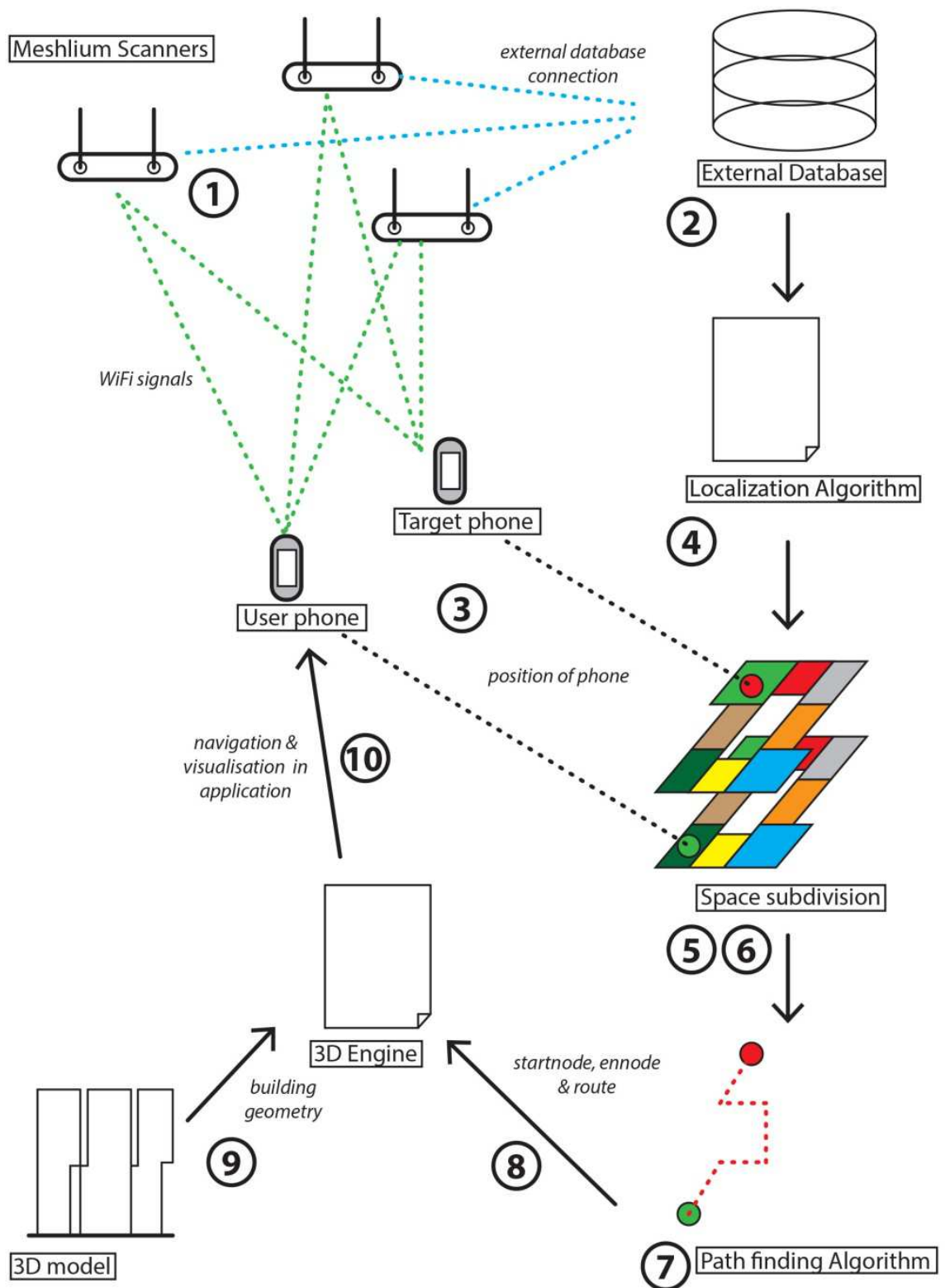


Fig. 8. Visualization of the system concept

## 5.2. CONCEPT SYSTEM ARCHITECTURE

The application 'Catch-a-Colleague' consists of several components and should provide various functions. First and foremost, the application should localize an employee. This localizing is done using Wi-Fi monitoring. A localization algorithm determines the position of the user and the employee that he/she wants to find. When the employee is found, the user wants to be navigated to the colleague. A path finding algorithm takes the positions of the user and the employee and calculates the best route to take. The best route depends on the shortest path, or for example minimum costs. This route will be visualized by a rendering program and made visible on the smartphone. The three main functions of the system are thus:

- Localization: determines the location of the user and the colleague they want to find
- Navigation: determines the best route to take from the location of the user to the employee
- Visualization: shows the best route in 2D, 2.5D or 3D

An app by itself consists of the following:

- System language or the programming language
- Operating System (OS)
- If necessary: Database Management System (DBMS)
- If necessary: a Web Service

The Operating system for the smartphone application will be Android, as requested by the Municipality of Rotterdam. The programming language will be Java/JavaScript and Python. For this application that requires a lot of data, a DBMS is necessary, to keep the data organized in one place and to be able to query fast.

The database can consist of the following data:

- Data about the 3D- model (floors/walls/doors/vertical elements, such as stairs and elevators)
- Data for connectivity (the chosen network or grid, with coordinates related to a space subdivision)
- Data from the scanners ( at least ID, timestamp, MAC, Access Point, RSSI, Vendor)
- Data about the employees (name, MAC-address, department)

The team prefers, if the team has a 3D- model in time, to store the geometry in the database. This is aiming to have all the data in the database and to easily request the right parts of the geometry for the visualization. This will lead to an application that is light.

The tables stored in the database contain an employees table, which links each employee to the relevant MAC- address. When an employee's location is requested, the corresponding MAC-address can be selected. The entries for this MAC-address of the latest 5 minutes will be selected by the localization algorithm. The localization algorithm will appoint the employee to one subspace or cell, linked to a floor level. The same process will be repeated for the user employee's location. The locations can then be linked to a network, consisting of nodes and edges for each floor. Then a path finding algorithm can be run to find the best route to take from the user to the searched employee. Based on the data from this route calculation, navigation can be given to the user and the right parts of the 3D geometry can be selected for the visualization.

If the 3D model was provided, the team would have liked to use the PostGres/PostGIS database, because spatial objects could then be stored. Instead, the team has used the MySQL database, which suits better for the simpler relational purposes of the project and efficient querying.

The database can be stored on an external server or locally stored on the smartphone. If the large amounts of data would be stored locally, the application would be slow and heavy to run, because the storage of data will require a lot of memory. Therefore the data will be stored in an external DBMS, which also has the advantage that updating needs to be done on only one place, instead of updating the locally stored data on each phone. Also ensures the external DBMS a more safe application, since you need to log in to have access to the data, in comparison to the fact that everyone can access the locally stored data on the phone.

In Figure 9 the general difference is represented.

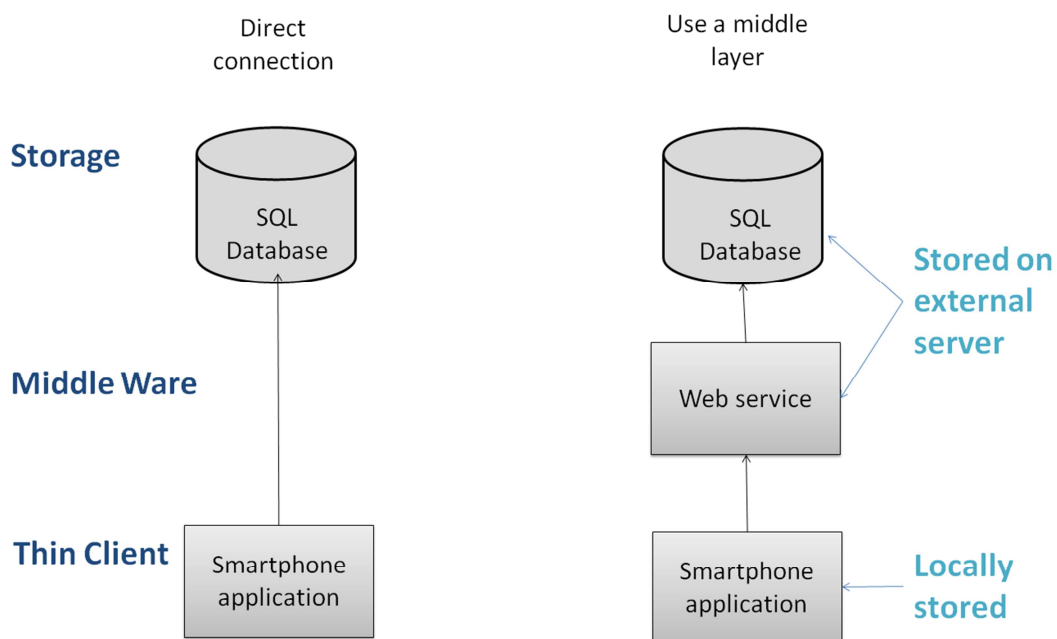
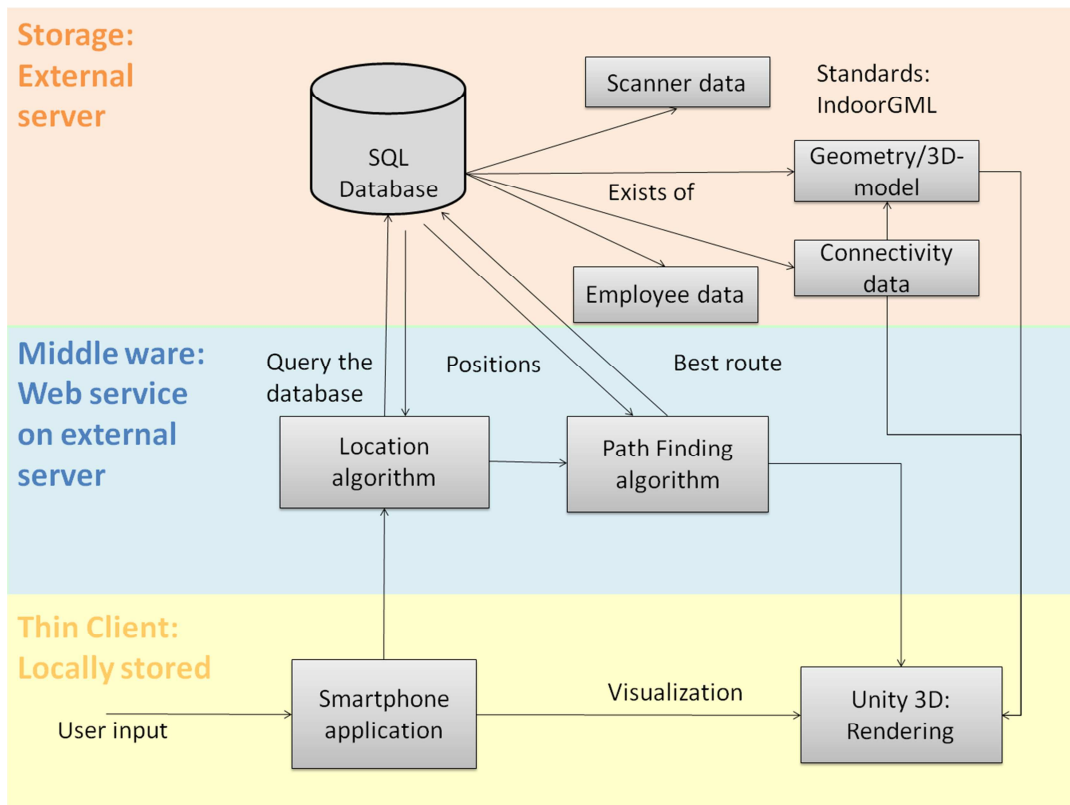


Fig.9. Direct connection or connection through a middle layer

A web service, also stored externally on a server, can act as a middle layer between the client and the database. The calculations on the database can be done by the web service, which calls the files that contain the algorithms and gets the results back. This service is not necessary, since this calling of files can be done on the phone too, but this will cost more memory which can result in the app to run slowly. This lead to the constrained options the team can explore to try to implement a web service.

The concept System Architecture for the Rotterdam Synthesis project looks as follows (Figure 10):



*Fig. 10. Concept System Architecture*

To be able to create an application that is light and runs smoothly as was one of the requirements, it would be the best if the phone itself will be seen as a thin client, thus only display of the route in a model will be done on the phone and that a web service, which consists of the calling of the algorithms written in Python code, would be used. The actual implementation of the components in the application is further described in chapter 11.

## 6. Literature research and theory

This chapter describes the literature study the team has done to create the several components and the prototype. First the report describes an important aspect of the application, namely privacy. Then the separate components are described from literature.

### 6.1. PRIVACY

In the process of the development of the ‘Catch-a-Colleague’ application, privacy is important since some issues related to the use of the users location information may arise. The ‘Catch-a-Colleague’ application can be considered as part of the category of Location-Based Services (LBS), namely services delivered according to the location of the user. The problem of protecting user’s privacy in Location-Based Services has been extensively studied, since the quick development of the latter in recent years and since privacy issues may affect the success of the services themselves (Bettini et al., 2009).

In The Netherlands the right to privacy is based on the Dutch constitution in article 10, but also European legislation provides an additional basis in the protection of privacy in The Netherlands (van Loenen et al., 2008). According to the Data Protection Act (WBP, 2000), which implements the Directive 46/95/EC into Dutch legislation, personal data (data on individual person) may only be processed for specified and legitimate purposes and no longer stored than strictly necessary (van Loenen et al., 2008).

In addition to the general privacy legislation, privacy and processing of personal data in an employment relationship have to be considered in the case of the ‘Catch-a-Colleague’ application. In fact, some questions need to be investigated: How do the privacy legislations apply in a working sphere? Can an employee trust on privacy during working time, when using devices from his employer?

In an employee-employer relationship it can be justifiable for an employer to check the e-mail and internet use of his employees. The Dutch Data Protection Authority has published a report, “Working well in networks”, in which guidelines are provided on how to check the e-mail of individual employees. In the Netherlands there are a lot of cases concerning Internet and e-mail monitoring and camera surveillance in the workplace, but so far there are only few cases concerning localisation of employees. However, from the few cases, it can be concluded that the same reasoning will apply as is the case with regard to internet, email and camera surveillance. At least there has to be knowledge by the employee that he can be monitored or watched (FIDIS, 2009).

Furthermore, according to the Telecommunications Act (Telecommunicatiewet, 2012) necessity of the processing of location data requires to provide a value added service. In the occasion of mere monitoring of employees, there is in fact no value added service, so in general this way of monitoring is prohibited, unless there is a prior informed consent of the individual data subjects (FIDIS, 2009).

In the case of the ‘Catch-a-Colleague’ application, smartphones have been provided to the employees by the Municipality of Rotterdam, thus many privacy issues are not occurring in this case. In fact, if it were personal phones of the employees, many privacy concerns would take place and therefore the data would fall under the data protection regulation. However, the application should still have a ‘switch off’ button, which allows the employee to disable the localization service when he/she doesn’t want to be located, when for instance the employee is not sitting at the desk and he/she is somewhere else (e.g. in the toilet). In addition, another option to avoid privacy concerns should be the creation of ‘groups of employees’, in such a



way that personal data is accessible only to certain colleagues, for instance to the ones of the same department.

The processing of personal data requires to provide value added service according to the Telecommunications Act, as previously mentioned, and in the case of the ‘Catch-a-Colleague’ application employee personal data is used to allow the localization of a colleague in a dynamic working space environment.

In developing the application, as the Dutch Data Protection Act states, personal data related to the employees (name, surname, department, MAC address etc.) should be stored in the database no longer than strictly necessary and processed only for the localization purpose of the application, in order to guarantee the privacy of the employees.

Summing up, the employee's permission should be asked in advance, the purpose of collecting the MAC address of the devices should be clearly stated and personal data should be sufficiently protected from unauthorized use.

## 6.2. SPACE SUBDIVISION

Different spatial models can be chosen to model the indoor environment: geometric models for representing the shape and the metric properties of spatial objects, topological models to highlight the relation between spatial objects, whereas semantic models to focus on the meaning of spatial features. Depending on the application, all of these models can be combined and hybrid models may be developed. The following scheme (Figure 11) summarizes the main indoor spatial models.

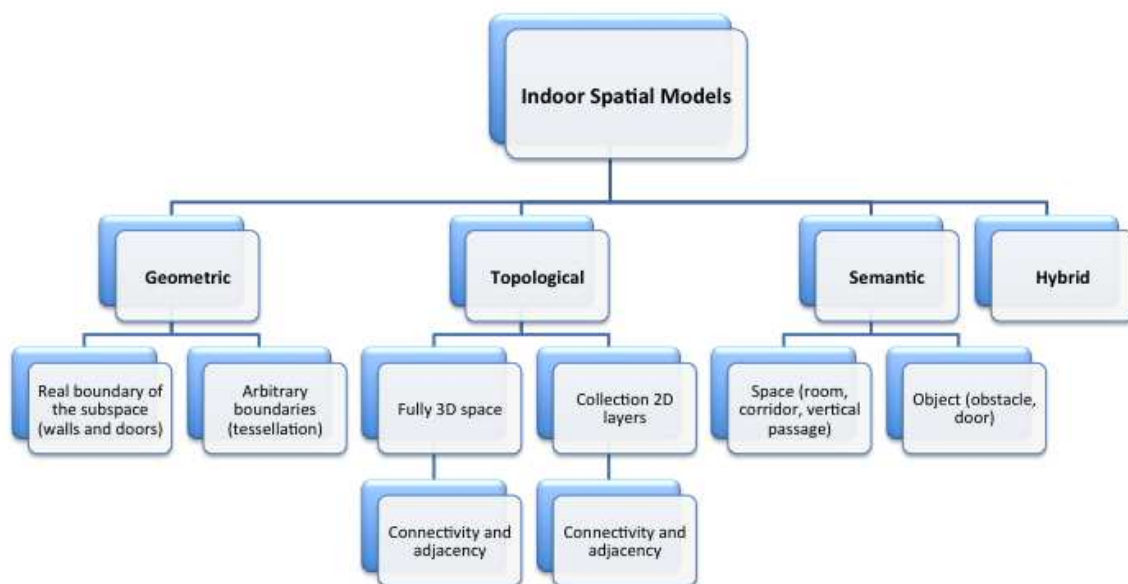


Fig.11. Indoor Spatial Models.

### 6.2.1 Standard IndoorGML

For the implementation of the space subdivision, standards for indoor navigation have been taken into account since they are crucial to ensure the compatibility and interoperability of indoor spatial information. Several standards such as CityGML, KML, and IFC have been published to describe 3D geometry and semantics of buildings, but they lack important features that are required by indoor navigation applications.

In this project, Indoor GML, a candidate OGC standard, has been considered since it provides a common framework of representation and exchange of indoor spatial information, especially for indoor LBS and routing services.

In IndoorGML, an indoor space is defined as a set of cells (cellular space) with an identifier (ID) and a certain location (x,y,z coordinates). Indoor space may also contain additional information: semantics, geometry and topology.

Semantics is used to classify and identify a cell and to determine the connectivity between cells. For instance, one of the most commonly used classification of cells is into navigable (rooms, corridors, doors) and non-navigable (walls, obstacles) cells. This classification is useful to navigate through cells (connectivity), since to be able to go from one room to another, the knowledge that at least one common opening (door, window) cell exists (OGC IndoorGML, 2014).

The geometric representation of indoor space is not a major focus of IndoorGML, since they are clearly defined by other standards like ISO 19107, CityGML, and IFC. However, it is still possible to represent geometry in IndoorGML (see Figure 12):

- using external links to objects defined in other datasets (e.g. CityGML)
- including geometry within a IndoorGML document
- including no geometry within a IndoorGML document

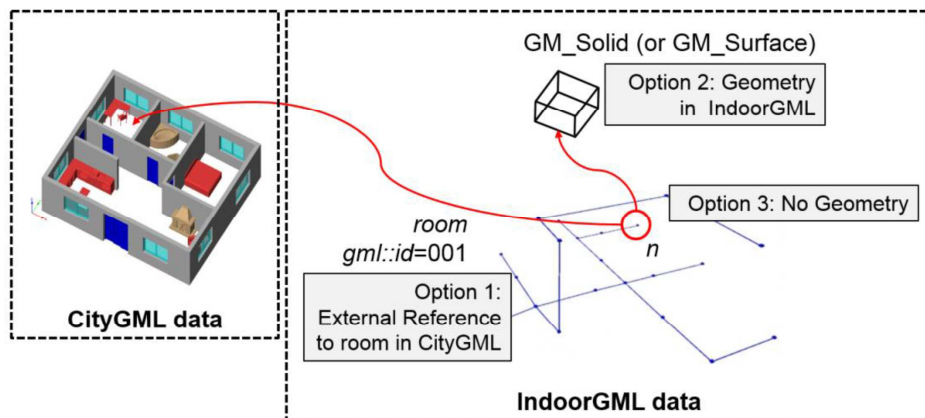


Fig.12. Geometric representation of indoor space. (Source: OGC IndoorGML, 2014).

Topology is an essential component of IndoorGML and topological relationships among indoor objects (e.g. adjacency and connectivity) are explicitly described with the Node-Relation Graph (NRG). Once adjacency relationships between cells are determined by Poincaré duality, other topological relationships can be defined from adjacency-relationships-based semantic information (OGC IndoorGML, 2014) (see Figure 13).

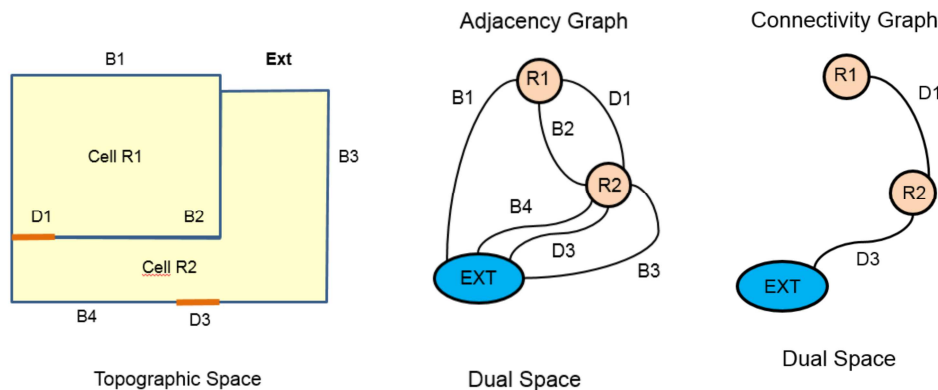


Fig.13. Adjacency and connectivity graph. (Source: OGC IndoorGML, 2014)

In IndoorGML, besides the geometric, semantics and topological models, another way of representing the space is defined: the Multi-Layered Space Model (MLSM), which supports multiple representation layers with different cellular spaces. According to this model, the same indoor space can be represented for instance as a topographic space, composed of rooms, corridors, and stairs, but also as different spaces with WiFi coverage cells and RFID sensor coverage cells (OGC IndoorGML, 2014) (see Figure 14).

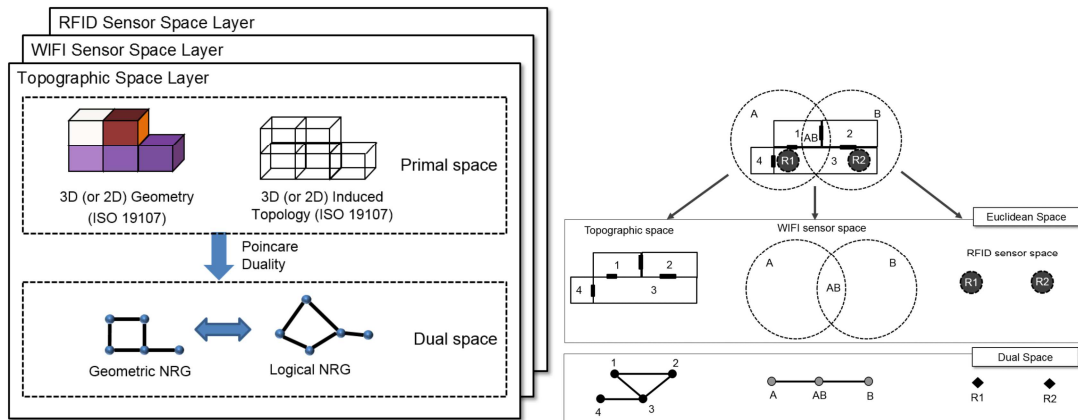


Fig.14. Multi-Layered Space Model (left) and an example of a Multi-Layered space representation (right). (Source: OGC IndoorGML, 2014)

### 6.3. LOCALIZATION

For the purpose of the designed application, the localization part is the most important and therefore part of the research question. Extended research on localization techniques was performed, based on literature and case studies. The general view is that most studies are focused on their specific case uses or empirical models and scientific formulas which when applied to the research environment have varying results. Promising new methodologies seem to make use of models which could better generalize and make a base abstraction of the problem, but at the same time remaining dependent on the real world environment and surroundings in every case.

#### 6.3.1 Scanner Placement

Scanner placement is a problem to be solved on its own, assuming all hardware is of identical absolute capabilities. The local environment of walls, material, noise sources and human intervention plays an important role which has great impact on signal strength, RSSI values, multipath, the Fresnel zone effect (Zomax Wireless, White Paper, 2010) and Wi-Fi coverage.

#### 6.3.2 Research

Some research has been performed on this area, which still remains a questionable problem for wireless network vendors and systems engineers. Below a number of considerations are shortly presented, but due to the extent of this project, both in time and applicability, they are not assessed nor further researched.

- In (J.Mulligan, 1997) various **attenuation factors** are considered and translated into scientific formulas before tested out for performance. The research case is based on a

generalized view of the problem and it is considered not to apply firmly on every case, thus the present one. The author concludes: “Thus far, researchers have not found a large scale path loss model which closely matches measurements within homes. This may be an indication that new parameters need to be introduced into the path loss model, such as construction materials and layout of the home.”

- Specifically for triangulation-based localization, an automated way of defining the placement of sensors into a known environment is researched in (O.Tekdas, 2010). A number of algorithms are presented but in the end, the problem is NP-complete, which is a hindering factor in applicability.
- Empirical models based on **ray tracing** are taken into consideration in the creation of (Winprop, 2010). The models are deterministic and therefore require specifications on path loss exponents and attenuation factors of underlying building materials. 3 models are considered (see Figure 15):

- i. One Slope Model (only path loss exponent)
- ii. Motley Keenan Model (adding the intersection with walls)
- iii. COST 231 Multi-Wall Model (individual material properties per wall)

In addition, multiple floor coverage is taken into account in the overall solution.

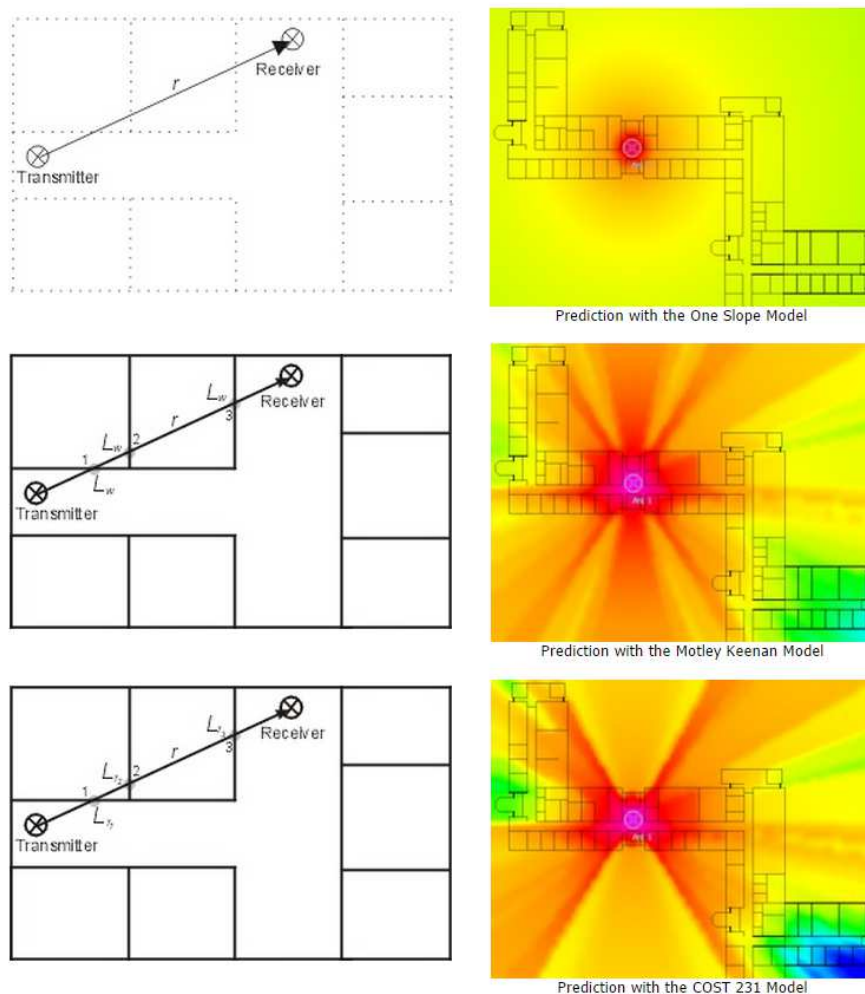
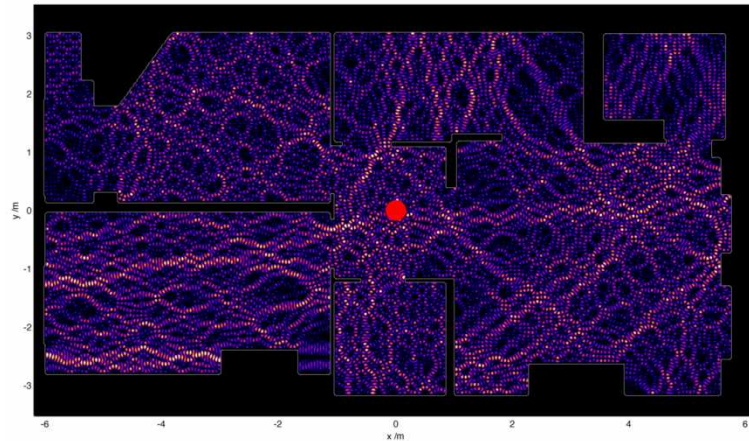


Fig.15. Different empirical models for signal coverage within a building.

- Promising recent research was performed by (J.Cole, 2014) making use of the **Helmholtz equations**, which assume a solution independent of time. When the latter factor is considered and a Finite Difference Time Domain (FDTD) technique is carried out, the resulting outcome is an oscillating field view of the coverage area (view [here](#)). In Figure 16, a static view of the field created justifies the positioning of the device in the middle of the environment, which was the purpose of the author’s research.



*Fig.16. Electromagnetic intensity map making use of the Helmholtz equations.*

All of the above could have had a positive influence in achieving the best possible outcome for the current project. Unfortunately, the scope of the research, time restrictions, background knowledge of the team but also the uncertainty of translating obstruction materials into coefficient factors does not allow for making use of such models.

### 6.3.3 Case Study Scanner Placement

Although the previous part constitutes an area of scientific research by itself, a more intuitive approach was performed in the current case study. Considering the number of devices at hand (4 Meshlium scanners) and the case environment (‘De Rotterdam’ building), a number of possible layouts was designed, for which the application could be tested. In the end four were chosen which were considered the most useful ones to test the capabilities of the scanners and use in the final testing of the application.

Therefore, in Figure 17 the following layouts are presented:

- *Max distance*: For this layout the four corners of the building were chosen. The reasoning behind it was that each scanner can selectively cover about ¼ of the floor area with “good” signal strength. Thus it would be easier to detect the user in the working areas, whereas the central area (which is mostly for walking, moving around) are still covered but with medium signal strength. By combining readings from multiple scanners these areas could also be distinguished.
- *Concrete Square*: Scanners are placed close to the corners of the central empty-space rectangle which includes the elevator area, stairs and non-working areas. This placement brings scanners closer to allow for maximum coverage, while at the same time keeps a high possibility for distinctions in the signal strength due to the thick walls in each corner.

- *Elevator & pathways*: Similar to the above, this placement would normally allow for maximum coverage, but this time the pathways and the elevator area are better covered, possibly allowing for a distinction by similar signal values per scanner couple.
- *Half-building*: In this final case, the desired option to check is what the outcome would be if using a denser population of scanners per floor. That is to search how the algorithms perform in a smaller area of the building, and whether it is better to have more than 4 scanners per floor.

Each of these layouts has its own purpose as far as floor coverage is concerned. Taking into consideration the physical environment and the scanner range, the two first methods seemed to be the most appropriate. Thus, the testing was focused on them as it will be described later.

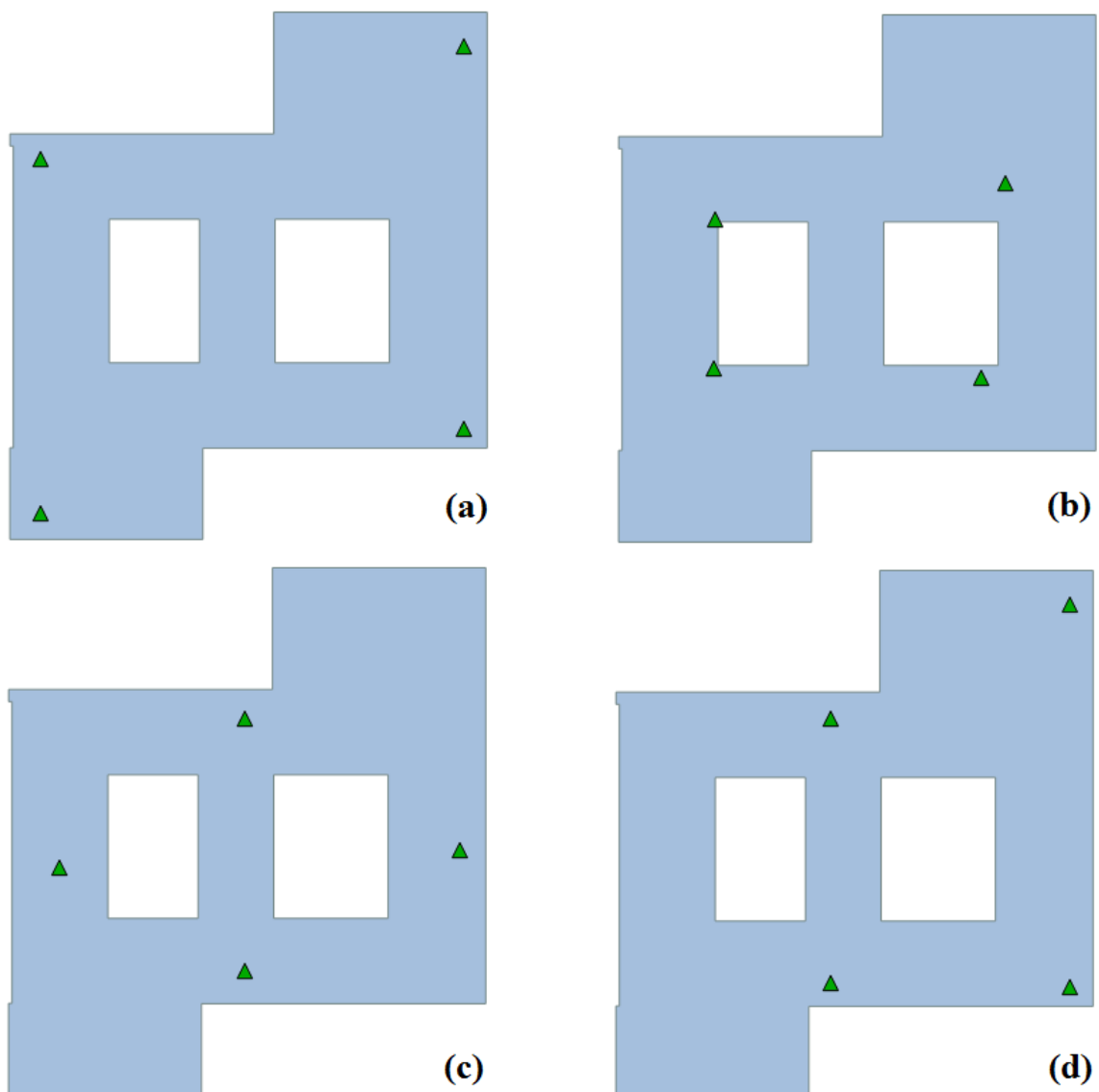


Fig.17. Scanner placement: (a) Max Distance, (b) Concrete Square, (c) Elevator & pathways, (d) Half-building, no thick wall interference

#### 6.3.4 Localization Methodologies

Indoors positioning systems (IPS) are systems which utilize sensors (in our case, meshlium scanners) in order to locate the position of a specific electronic device like smartphones. A regularly visited option is Wi-Fi fingerprinting [(Wikipedia: Wi-Fi Positioning System,2014), (M.Quan,2010), (V.Moghtadaiee,2014), (W.Ching,2010)]. The research value in the current project's case is that it tries not to follow the norm and explore other areas as well.

#### 6.3.5 Theoretical Background

Given the current project's context and the hardware provided, the main measurements that can be utilized for localization are the RSSI values received by the scanners. RSSI stands for **Received Signal Strength Indicator** and can be a rough decision factor of proximity for a scanned device to a transmitter antenna. On this topic, research has been performed on the actual relation between these values and their possible translation to distance from transmitter antennas.

In (K. Benkič, 2008), given certain hardware support and 3 different models (free space, two-ray, log-distance) RSSI values together with a Link Quality Indicator (LQI) were utilized to find the proximity and accuracy of readings. The results according to the authors depend on the hardware functionality, environment and actual goals of the application of how accurate one achieves to be.

A multimodal approach is followed in (E.Martin, 2010) where Wi-Fi, cellular communications radio and accelerometer of smartphones are integrated to provide an "accurate" localization measurement in room-resolution (up to 87% rate of success).

An interesting model is built in (A.S.Paul, 2008) where RSSI calibration data assume an observation function by fitting nonlinear maps between known calibration locations and RSSI mean values. A Bayesian framework of sigma-point Kalman filters (SPKF) incorporates the RSSI maps that fuse all sensor measurements with a simple dynamic model of walking. The dynamic model consists of a random walk model augmented with repulsive forces to account for room-wall reflections and attenuations.

#### 6.3.6 Localization Concepts

From the above research, it is understood that the underlying problem is dependent on various factors impeding its generalization. Furthermore, most applications aim to an empirical and "trial and error" solution that best fits each test case, therefore there is no general guideline to follow for the current project.

As such, in this case study four approaches were considered:

- a. an Area Rings Approach
- b. Multi-/Tri-lateration
- c. Triangulation
- d. WiFi Fingerprinting

For simplicity in the above cases, and without the degradation of generalization, a 2D environment of point-like devices and positions can be safely assumed.

##### *a) Area Rings Approach*

This approach provides a generic deterministic model of signal disintegration as distance increases between tracked device and scanner. It assumes an open field view of the scanners' radio signal intensity with no modeled interruptions (walls, multipath, noise etc). Each scanner defines a number of "rings" which each take up a range of RSSI values related to the

ones received from tests. The boundaries between the rings are considered to be fuzzy and each ring's thickness is assumed to increase the further away from the transmitter (Figure 18).



*Fig.18. Concept of area rings of signal intensity per scanner – no degradation.*

The rings of different scanners interleaving is not seen as an impeding factor but as a way to better define the area of localization. If multiple areas are perceived as feasible locations, the ones with the closest rings to scanners' radii are considered to be the most probable and chosen as correct.

This model is deterministic in its approach but requires a testing phase in order to set the values for each ring. A further step would be to try to model the field based on the environment interruptions the scanners face due to their placement and by that create not circular areas, but areas of varying shape.

#### *b) Tri-/Multi-Lateration*

By constructing a multi-metric function that mainly takes into account RSSI values and roughly translates them into distances of the devices from scanners, a trilateration method can be conceived. Trilateration is the process of determining a relative unknown position of the device at question, using the geometry of spheres or circles, whose radii are described by the above function (see Figure 19). It is a well-known method and has been used in surveying, navigation and the Global Positioning System (GPS). In the current case the main issue would be finding a suitable function for the translation of RSSI into distance.



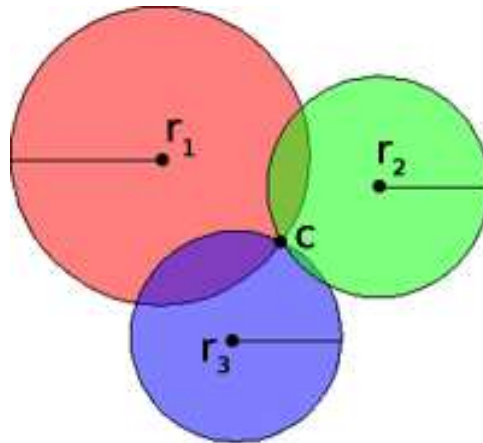


Fig.19. Trilateration (2D).

The problem of defining such a well-structured function is a matter of further research, but some considerations can be taken at hand. Considering a finite amount of time passed (enough to achieve a considerable accuracy), the function depends on the RSSI values collected and averaged (by a certain weight), a model of the surrounding environment (*env*) and a scanning factor which relates to the number of times a particular device was scanned (*s*). The model for the environment, **env** is a complex notion, since it involves attenuation factors, multipath and wall surface materials that relate to the physical world. The **s** factor can be a metric of “trustability”, given the max number of scans that can be performed in a certain time frame and the actual scans perceived.

Thus, the radius **r** outputted from that function, comes with an added error **r<sub>e</sub>** which increases as **r** increases as well (the further away the device is considered to be from the scanner, the lowest the credibility of the reading). The above function can be formulated as:

$$f(RSSI, env, s) = r \pm r_e$$

If the device is detected by 3 scanners, a position of the device can be estimated. Each scanner forms a “ring” of possible location area of the device. The intersection of 3 of those rings is enough to define an area that is accurate enough for the current project to localize a person in a sub-division of a room (see Figure 20).

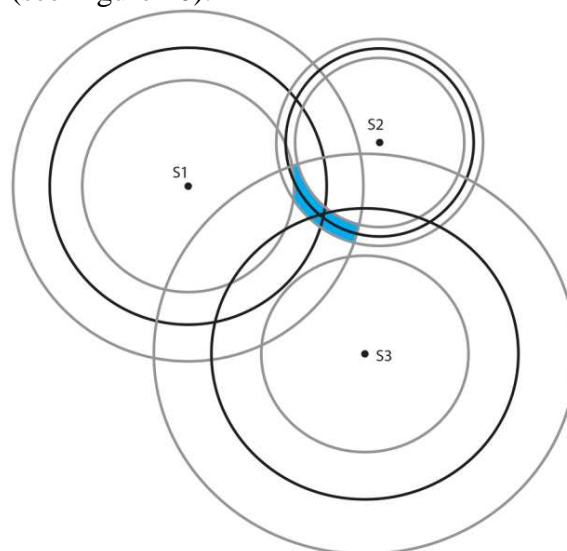


Fig. 20. Trilateration with error in radii included for each scanner (S1,S2,S3).  
The blue area is the most probable location of the device.

*Multilateration* is similar to trilateration but it uses more than 3 fixed points (see Figure 21). The multiple areas that are formed in this case may be assessed by a factor of “potentiality” that takes into account the  $s$  factor from previously and the number of scanners in the area, in relation to how many of them actually detected the device. By calculating this, a most probable area can be found to localize the device.

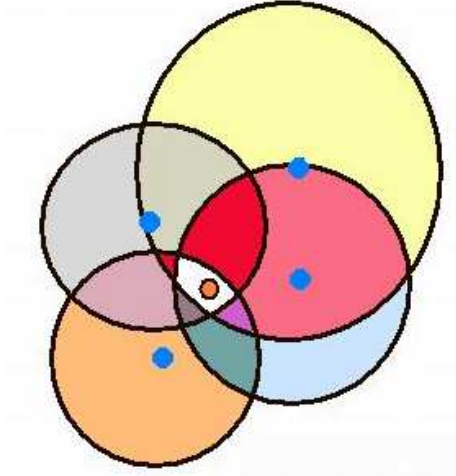


Fig. 21. Multilateration

c) *Triangulation (forward intersection)*

Triangulation is a well-known method using fixed points of known positions to calculate a third point’s position. In forward intersection, instead of using angles which is the usual method for triangulation, a calculated distance is used. Therefore, for this method, a function similar to the one described for trilateration above can be used, that accurately describes a triangle for every pair of scanners and unknown device location. Making this assumption of calculating distance, the position can be determined by the procedure that follows.

In more detail, for a triangle ABP (see Figure 22) point A and point B are fixed points with known coordinates. The lengths of its three sides are respectively a, b and c.  $\alpha$  and  $\beta$  are respectively the angle between sides b and c and the angle between side a and c. Point P is the unknown point whose location we would like to know.

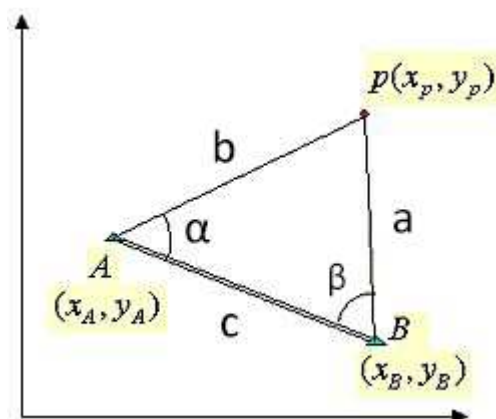


Fig.22. Triangle ABP

Then the coordinates of point P can be calculated according to the equations below.

$$x_p = \frac{x_A \cot \angle B + x_B \cot \angle A - (y_A - y_B)}{\cot \angle A + \cot \angle B} \quad (1)$$

$$y_p = \frac{y_A \cot \angle B + y_B \cot \angle A - (x_A - x_B)}{\cot \angle A + \cot \angle B} \quad (2)$$

If we draw a perpendicular line  $h$  through point  $p$  to side  $c$ , side  $c$  would be divided into  $b_1$  and  $a_1$  (see Figure 23).

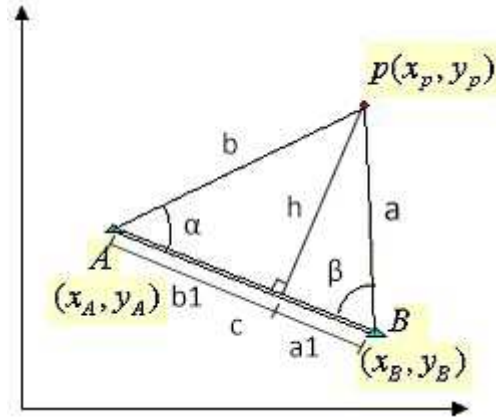


Fig. 23. Triangle ABP

Then

$$\cot \angle A = \frac{b_1}{h} \quad \cot \angle B = \frac{a_1}{h}$$

Take them back into equations (1) and (2), following equations then are obtained:

$$x_p = \frac{a_1 x_A + b_1 x_B - h(y_A - y_B)}{a_1 + b_1}$$

$$y_p = \frac{a_1 y_A + b_1 y_B + h(x_A - x_B)}{a_1 + b_1}$$

$$\text{Where } b_1 = \frac{c^2 + b^2 - a^2}{2c}, \quad a_1 = \frac{c^2 + a^2 - b^2}{2c}, \quad h = \sqrt{a^2 - a_1^2} = \sqrt{b^2 - b_1^2}$$

For 2 known points and distances, 2 possible positions may exist for the unknown point.

If more than 2 known positions exist, triangulation for each pair of points can be performed, thus a number of calculated points for the unknown are found. If  $n$  is the number of known points, the number of possible positions for the unknown complies with the formula:

$$N = n(n-1)$$

In example (Figure 24), if 3 points (scanners in our case) are known and distances from each of them have been calculated, 6 possible positions exist for the unknown point given a margin of error. In order to find the position that best fits, all possible combinations of those positions are considered ( $2^{N/2} = 2^{6/2} = 2^3 = 8$ ). By calculating the standard deviations, a best possible answer can be chosen.

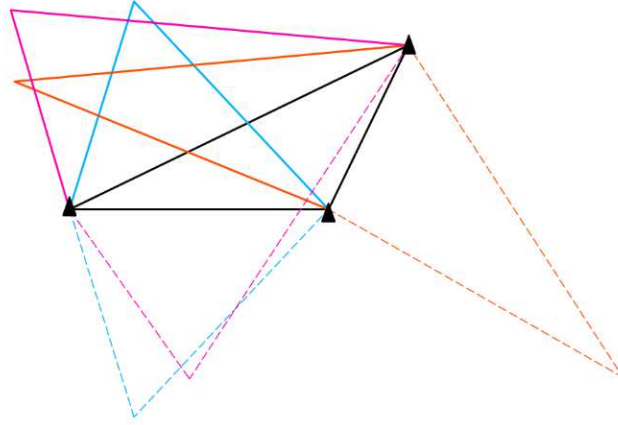


Fig. 24. Triangulation with 3 known points

#### - Accuracy Analysis

According to error propagation, the error of the outcome of a function can be estimated by the error of its inputs. In this case, the lengths of  $a$  and  $b$  are the inputs that are going to be measured by us based on the received RSSIs and the coordinates of point P is the final outcome. Therefore,

$$\begin{aligned}\delta_{x_p}^2 &= \left(\frac{\partial x_p}{\partial a}\right)^2 \delta_a^2 + \left(\frac{\partial x_p}{\partial b}\right)^2 \delta_b^2 \\ \delta_{y_p}^2 &= \left(\frac{\partial y_p}{\partial a}\right)^2 \delta_a^2 + \left(\frac{\partial y_p}{\partial b}\right)^2 \delta_b^2 \\ \delta_P &= \sqrt{\delta_{x_p}^2 + \delta_{y_p}^2}\end{aligned}$$

Where  $\delta_a$  and  $\delta_b$  denote respectively the error of input  $a$  and  $b$ ,  $\delta_{x_p}$  denotes the error of  $x$  coordinate of point P,  $\delta_{y_p}$  denotes the error of  $y$  coordinate of point P,  $\delta_P$  denotes P's overall point position error.

If  $\delta_a$  and  $\delta_b$  are assumed to be equal, i.e.  $\delta_a = \delta_b = \delta$ , then

$$\begin{aligned}\delta_{x_p}^2 &= \left[ \left(\frac{\partial x_p}{\partial a}\right)^2 + \left(\frac{\partial x_p}{\partial b}\right)^2 \right] \delta^2 \\ \delta_{y_p}^2 &= \left[ \left(\frac{\partial y_p}{\partial a}\right)^2 + \left(\frac{\partial y_p}{\partial b}\right)^2 \right] \delta^2\end{aligned}$$

Where

$$\begin{aligned}\left(\frac{\partial x_p}{\partial a}\right)^2 + \left(\frac{\partial x_p}{\partial b}\right)^2 &= \frac{a^2 + b^2}{c^4} (x_A - x_B)^2 + \frac{(y_A - y_B)^2 \Delta_2}{c^4 \Delta_1} + \frac{(x_A - x_B)(y_A - y_B) \Delta_3}{2c^4 \sqrt{\Delta_1}} \\ \left(\frac{\partial y_p}{\partial a}\right)^2 + \left(\frac{\partial y_p}{\partial b}\right)^2 &= \frac{a^2 + b^2}{c^4} (y_A - y_B)^2 + \frac{(x_A - x_B)^2 \Delta_2}{c^4 \Delta_1} + \frac{(x_A - x_B)(y_A - y_B) \Delta_4}{2c^4 \sqrt{\Delta_1}}\end{aligned}$$

And

$$\Delta_1 = -a^4 - b^4 - c^4 + 2a^2b^2 + 2a^2c^2 + 2b^2c^2$$

$$\Delta_2 = \frac{1}{16} \left[ \left( \frac{\partial \Delta_1}{\partial a} \right)^2 + \left( \frac{\partial \Delta_1}{\partial b} \right)^2 \right] = a^6 + b^6 + 4a^2b^2c^2 + (c^4 - a^2b^2)(a^2 + b^2) - 2c^2(a^4 + b^4)$$

$$\Delta_3 = -a \frac{\partial \Delta_1}{\partial a} + b \frac{\partial \Delta_1}{\partial b} = 4(a^2 + b^2 - c^2)(a^2 - b^2)$$

$$\Delta_4 = a \frac{\partial \Delta_1}{\partial a} + b \frac{\partial \Delta_1}{\partial b} = -4(a^4 + b^4 - 2a^2b^2 - a^2c^2 - b^2c^2)$$

A simple example is shown in Figure 25. Just for the convenience of calculation, the triangle is assumed to be equilateral and its known side  $c$  on the bisector of the angle between  $x$  and  $y$  axes.

Thus

$$x_A - x_B = y_A - y_B = \frac{c}{\sqrt{2}}$$

Besides, the error of the measured distance is assumed to be constant and not to change with the distance. Then, the table below is derived showing in this case how the error of the measured distance affects the final point position accuracy.

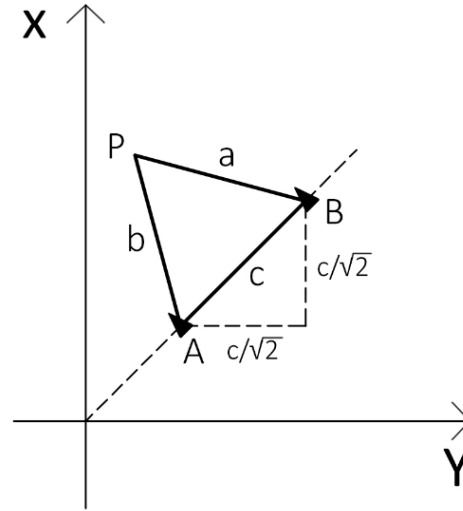


Fig. 25. Example error calculation

$\delta(m)$	$\delta_{x_p}^2(m)$	$\delta_{y_p}^2(m)$	$\delta_p(m^2)$
0.1	0.115470053838	0.157735026919	0.195483175876
0.5	0.57735026919	0.788675134595	0.977415879379
1	1.15470053838	1.57735026919	1.954831758762
2	2.30940107676	3.15470053838	3.90966351751
3	3.46410161514	4.73205080757	5.86449527627
4	4.61880215352	6.30940107676	7.81932703503
6	6.92820323028	9.46410161514	11.7289905525
8	9.23760430703	12.6188021535	15.6386540701
10	11.5470053838	15.7735026919	19.5483175876
15	17.3205080757	23.6602540378	29.3224763814
20	23.0940107676	31.5470053838	39.0966351751
30	34.6410161514	47.3205080757	58.6449527627

Table 1. Point position accuracy resulted from different error of measured distance

#### d) Wi-Fi Fingerprinting

The process of Wi-Fi fingerprinting consists of two phases: the training phase and the matching phase. On the training phase, the given area is divided by a grid into many small cells, the size of which is decided by how big the area is and how accurate the localization is required for certain application. Each cell in theory has a unique profile of Received Signal Strength Indicator (RSSI) of the Wi-Fi access points (APs) (see Figure 26), which depends very much on the specific placement of the APs as well as the in-situ environment. Thus, a training database of these profiles at every cell needs to be established in advance to create heatmaps for each APs. However, collecting data at every cell is excessively time-consuming and this is where the interpolation of a few sampling points reasonably selected from the grid shall be introduced.

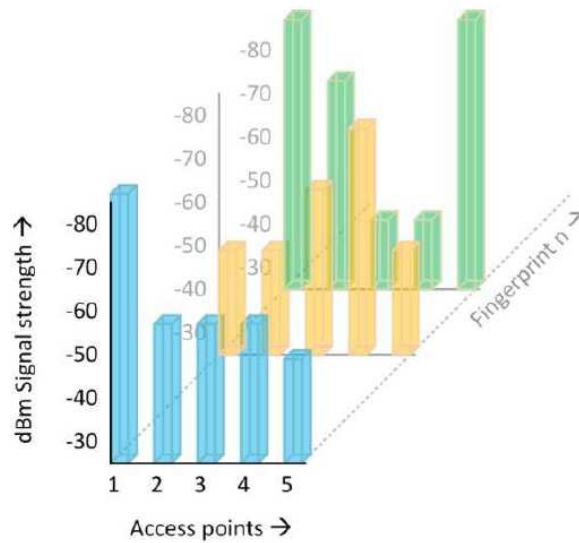


Fig. 26. Example of fingerprints: the blue, orange and green columns respectively indicate the fingerprints at three different locations. There are five RSSI values received from five access points in the fingerprint of each location. Each of these fingerprints presents a unique characteristic different from that of the others.

Then, live RSSIs can be compared with the fingerprints in the heatmaps using certain matching algorithm to find the best match with its coordinates (x, y) in a pre-set reference system. There are two matching methods used most commonly.

One is the least sum of squares, which is also called Nearest Neighbor. This method sums all squared differences between live signal strength with recorded signal strength per location and location with the least sum of squared differences then is assumed to be the actual location.

$$MIN \left( \sqrt{\sum_{i=1}^n (R_i - FP_i)^2} \right)$$

The other, counting within a search space, defines a matching range for each AP using a pre-set deviation  $\delta$  allowed for the signal strength. If recorded signal strength is  $FP_i$ , a match is

achieved when the live signal strength  $R_i$  is within  $[FP_i - \delta, FP_i + \delta]$ . Location with the most matches is assumed to be the most likely location (see Figure 27).

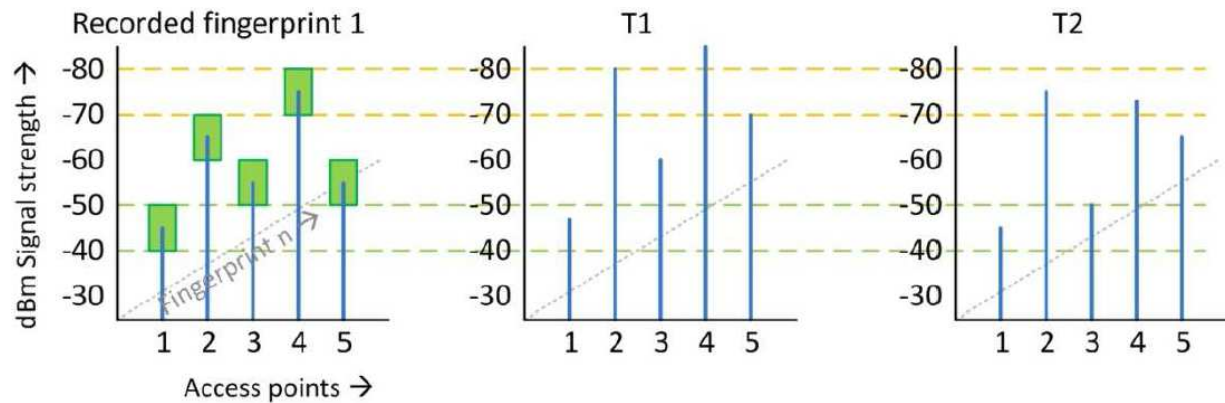


Fig.27. Example counting within a search space: each of the RSSI values received from the five different access points in this recorded fingerprint defines a green bucket indicating the range, within which a live RSSI value will be considered a match. In this case, T1 has two matches, while T2 has three matches.

### 6.3.7 Pros and Cons of Localization Methodologies

In the previous section 4 different localization methods was described that can be assumed for the study case. These methods are considerably different in approach especially for the application that is the goal of this project and the accuracy of positioning that the system would try to achieve. To make a choice of one or two of these methods a number of advantages and disadvantages can be listed down, which are summarized in the table below.

Methodology	Advantages	Disadvantages
Area Rings	<ul style="list-style-type: none"> <li>+ simplicity</li> <li>+ assumes an easy to use model</li> <li>+ deterministic</li> <li>+ can be applied in different environments</li> </ul>	<ul style="list-style-type: none"> <li>- great errors</li> <li>- requires a testing phase</li> <li>- needs a geometry database</li> <li>- error calculation is dubious</li> </ul>
Tri-/Multi-Lateration	<ul style="list-style-type: none"> <li>+ can be the most precise</li> <li>+ best-described model</li> <li>+ takes into account factors that diminish the accuracy</li> </ul>	<ul style="list-style-type: none"> <li>- difficult to formulate</li> <li>- requires geometry intersections</li> <li>- dependent on a good space subdivision</li> </ul>
Triangulation (FI)	<ul style="list-style-type: none"> <li>+ easy to script</li> <li>+ mathematical solution</li> <li>+ no need for extra information storage on the database</li> </ul>	<ul style="list-style-type: none"> <li>- assumes a perfect-case scenario of known distances</li> <li>- computationally costly</li> </ul>
Wi-Fi Fingerprinting	<ul style="list-style-type: none"> <li>+ Precise and close to real distribution of values</li> <li>+ Can be updated by user input</li> <li>+ Field-values: grid granularity is defined by the application's purpose</li> </ul>	<ul style="list-style-type: none"> <li>- Time-consuming to collect data</li> <li>- Non-flexible: Changes depending on scanner layout</li> <li>- Input data quality needs to be guaranteed</li> </ul>

Table 2. Pros and cons of localization methodologies

## 6.4. NAVIGATION

### 6.4.1 Grid and network approach

All applications need space subdivisions to be able to address the destination point in the best possible way (Zlatanova et al., 2014). Many different approaches to subdivide the space exist and below the two main groups of approaches are discussed in detail: the grid and the network.

- In **grid approaches** the environment is subdivided into cells that can have the same shape and size (regular) (see Figure 28) or that can differ in shape and size (irregular).

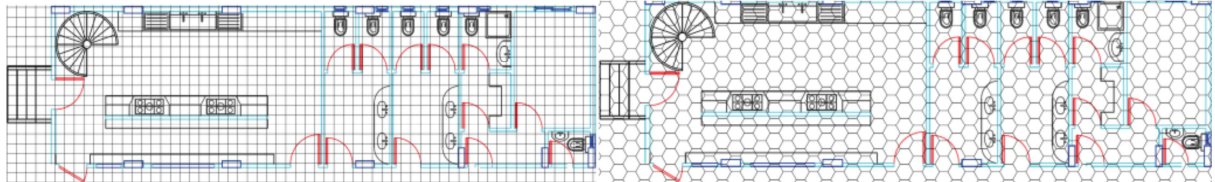


Fig. 28. Square and hexagon subdivision (Afyouni et al 2012).

In this approach a partition that covers the entire space is created, allowing very precise movement in the space. However, the accuracy of this method strictly depends on the cell size: in fact, if the grid is too coarse, important information might be lost, while if it is too fine it consumes a large amount of memory and processor time (Zlatanova et al., 2014) (examples shown in Figure 29).

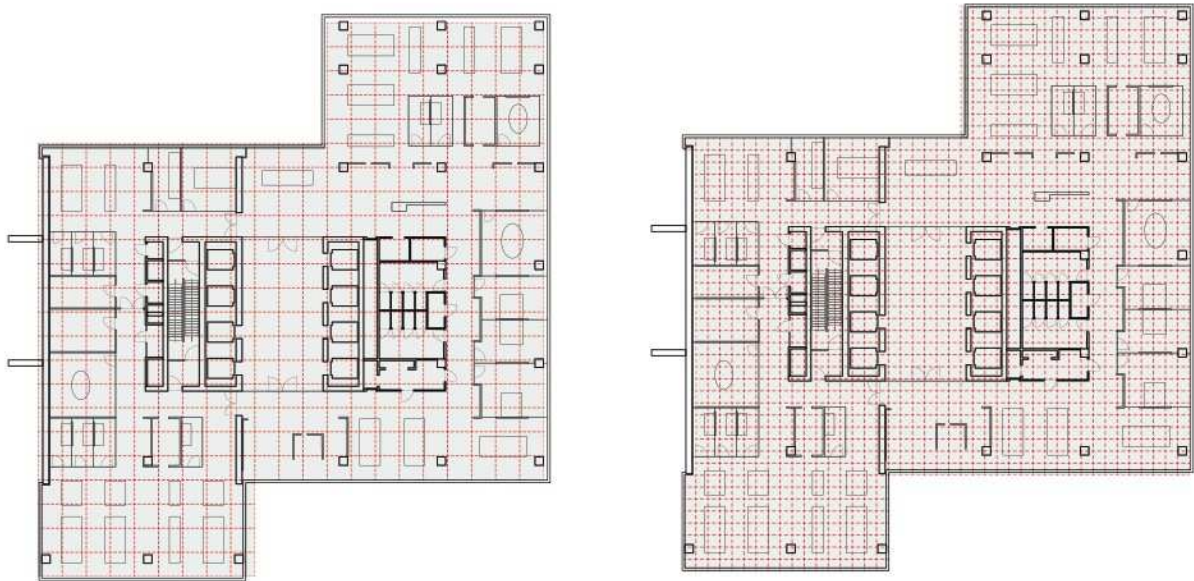


Fig.29. Regular grid space subdivision of 'De Rotterdam' building (2x2m on the left and 1x1m on the right).

- In **network approaches** topological-based structures are used to describe the connectivity and the adjacency of the different spatial units. In graph-based models the indoor space is represented as a graph where nodes model predefined locations and edges stand for the connectors (Afyouni et al., 2012).



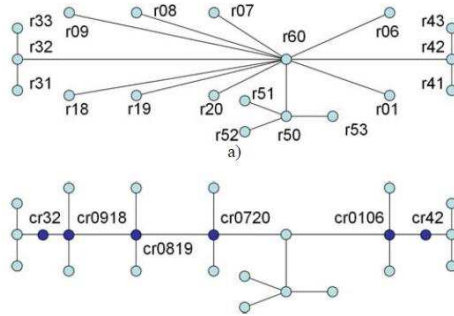


Fig. 30. 3D Geometric Network from 3D Topological Data Structure (Meijers, Zlatanova and Pfeifer).

The network can be designed manually or using a Voronoi diagram, Medial Axis Transformation, Poincaré Duality and visibility graph or a combination of them (Zlatanova et al., 2014). Poincaré Duality is often used to simplify the complex spatial relationships, mapping 3D solid objects in primal space, e.g. rooms within a building, to vertices in dual space. As it is shown in Figure 31, the common 2D face shared by two solid objects is transformed into an edge (1D) linking two vertices in dual space. Thus, edges of the dual graph represent adjacency and connectivity relationships which may correspond to doors, windows, or hatches between rooms in primal space.

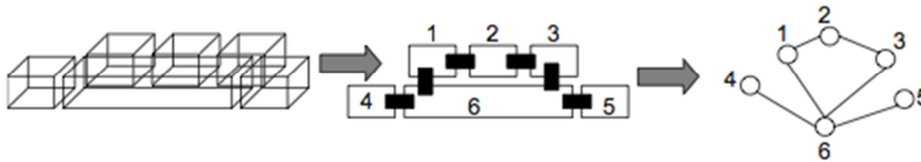


Fig. 31. Example for the partitioning of building interior into rooms and its representation in dual space.

The table below summarizes the pros and cons of grid and network approaches.

Grid model	Network model
<p><b>PROS:</b></p> <ul style="list-style-type: none"> <li>• Easy to build, represent and maintain</li> <li>• Accurate location description</li> <li>• High flexibility on granularity</li> <li>• Mainly used to represent continue phenomena</li> <li>• Used for robot navigation and games</li> <li>• Suitable for computation and for tracking</li> </ul>	<p><b>PROS:</b></p> <ul style="list-style-type: none"> <li>• Used for path finding applications</li> <li>• Based on Poincare duality (volume-&gt;node, surface-&gt;edge)</li> <li>• Information about obstacles</li> <li>• Mainly used for human navigation</li> <li>• Efficient because it is more compact</li> <li>• Good representation of connectivity</li> </ul>
<p><b>CONS:</b></p> <ul style="list-style-type: none"> <li>• Excessive amount of memory and processor time in large spaces</li> <li>• The size of the grid is critical: a too fine grid is computational expensive, a too coarse grid causes loss of information</li> </ul>	<p><b>CONS:</b></p> <ul style="list-style-type: none"> <li>• The location description can be inaccurate, if the network is very simple or course</li> </ul>

Table 3. Pros and cons of grid and network space subdivision models.

### 6.4.2 The network

According to Worboys and Duckham (2004), “A graph G is defined as a finite non-empty set of nodes, together with a set of unordered pairs of distinct nodes (called edges)”. In other words, a graph describes the locations of nodes and their interconnectivity. Different ways of storing a network graph are available, namely an Adjacency matrix, a set of labelled edges or an adjacency list. An example is shown below.

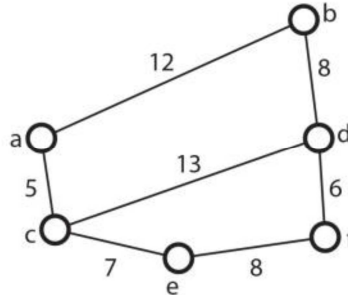


Fig.32. Example graph

The Adjacency matrix (see Figure 33) makes all data easily accessible for computational efficiency but is verbose. The adjacency list (see Figure 34) is more compact, but also enables easy computation for specific nodes. Finally the set of labelled edges (see Figure 35) uses the minimum amount of storage, but it is not as efficient to compute with.

	a	b	c	d	e	f
a	0	12	5			
b	12	0		8		
c	5		0	13	7	
d		8	13	0		6
e			7		0	6
f				6	8	0

Fig. 33. Adjacency matrix

a	(b,12),(c,5)
b	(a,12),(e,8)
c	(a,5),(d,7),(e,13)
d	(b,8),(c,13),(f,6)
e	(c,7),(f,8)
f	(d,8),(e,6)

Fig. 34. Adjacency list

{(ab,12),(ac,5),(bd,8),(cd,13),  
(ce,7),(cf,9),(df,6),(ef,8)}

Fig.35. Set of labelled edges

### 6.4.3 Path finding

Once the network is generated, a route computation is needed from the user's position towards the target position. When both positions have been appointed to a certain node, a computation of the shortest path between these nodes can be performed. The user's position will then be defined as a start node and the target's position will be defined as the end node. A path finding algorithm will search for adjacent nodes around the start node, until the end node has been found. The outcome of the path finding algorithm is a series of nodes, which connect the start node to the end node, with the shortest weight (in our case distance) possible.

Different path finding algorithms exist, varying on complexity and computational efficiency, each more suitable for a certain application. Since a starting point is already known, single source shortest paths algorithms are considered for a weighted and undirected graph. In our case the graph is undirected because any movement through the network can be done in both directions. The two most common path finding algorithm for this case are Dijkstra and A\*. These algorithms will be described below.

- The Dijkstra algorithm searches through adjacent nodes, based on a priority queue which extracts the nodes with a minimum weight (distance) from the source. This means that equal amounts of nodes are visited in all directions, until the end node has been found.

- The A\* algorithm is based on a goal directed graph-traversal strategy. This means it is searching in the direction of the target node. This is achieved by visiting nodes with the minimal sum of distance to the source node and the estimated distance to the target node (see Figure 36).

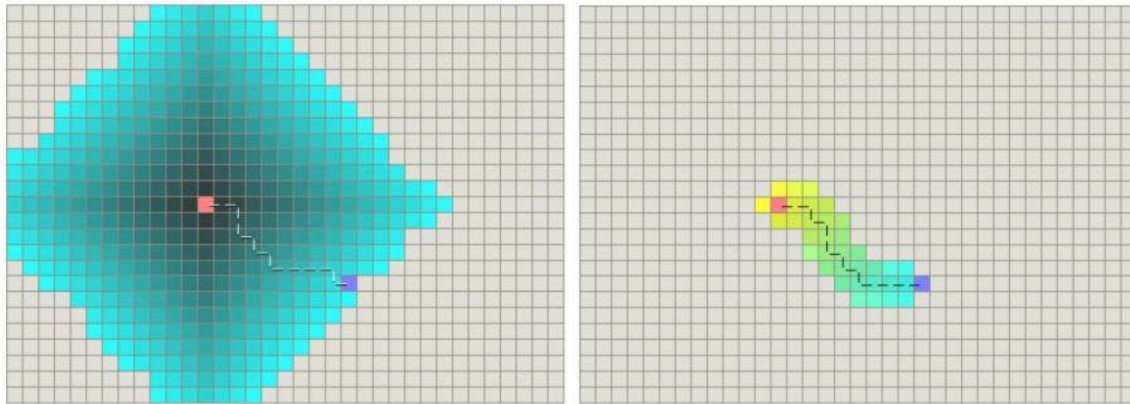


Fig. 36. Visualization of Dijkstra and A\* algorithm pathfinding on grid

#### 6.4.4 IndoorGML data model

As for the Space Subdivision component, the standard IndoorGML has been taken as a reference for implementing the navigation component. In this case, the IndoorGML data model has been considered, with its core module and thematic extension modules.

A SpaceLayer is an important component of the core module that represents each space layer, such as topography, sensor, security space, etc. In IndoorGML SpaceLayer aggregates State and Transition (example in Figure 37):

- State represents a node in dual space, which can be associated with a room, corridor, door, etc. within a building of the primal space. It is represented geometrically as Point in IndoorGML.
- Transition is an edge that represents the adjacency or connectivity relationships among nodes. Transition always connects two States.

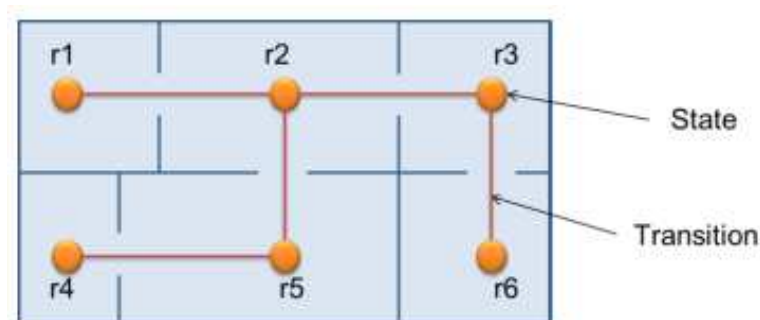
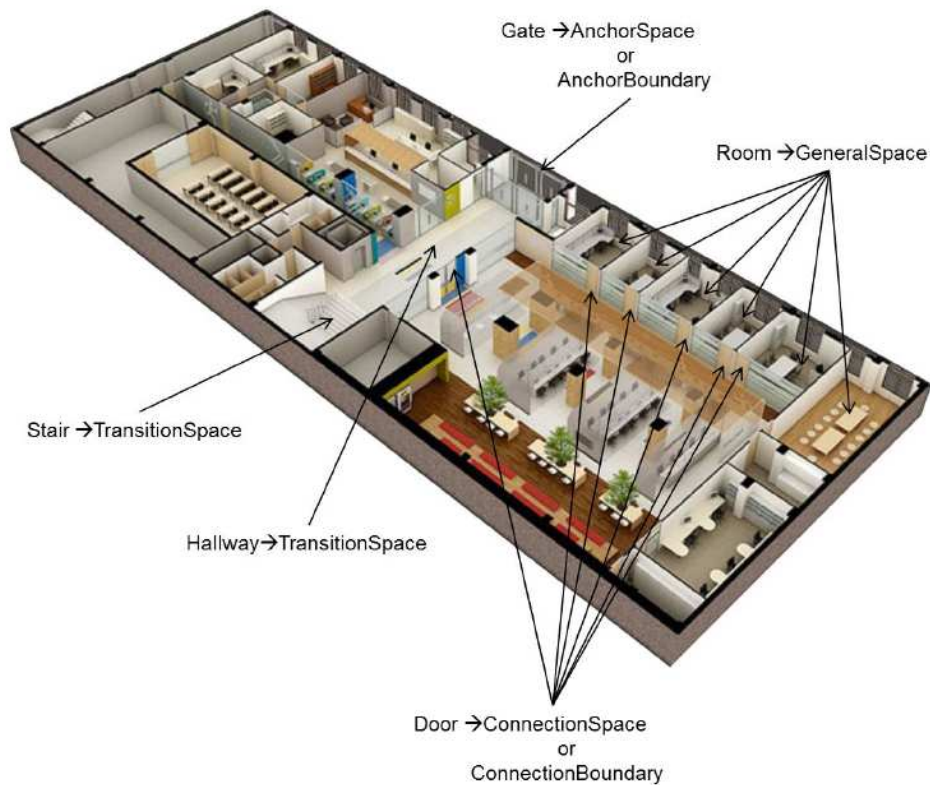


Fig.37. Example of Topographic SpaceLayer. (Source: OGC IndoorGML, 2014)

An important thematic module considered in modelling the space of `De Rotterdam` building is the IndoorGML Navigation Module, which specifies in detail the generic concepts of the core module, in the context of indoor navigation. According to this module, indoor space is represented by two classes:

- The NavigableSpace class, which denotes a space users can move freely in (e.g. compartmentalized spaces such as corridor, lobby, hallway, big room). The class has two subclasses GeneralSpace and TransferSpace (such as rooms, terraces, lobbies).
- The NonNavigableSpace class, which represents the space that is occupied by obstacles.



*Fig. 38. Indoor space mapped to IndoorGML Navigation module classes.  
(Source: OGC IndoorGML, 2014)*

## 7. The Libelium Meshlium Xtreme Scanners

### 7.1. INFORMATION ABOUT THE SCANNERS

The hardware used in this project consists of 4 Libelium Meshlium Xtreme Scanners, which are multiprotocol router for wireless sensor networks. The Meshlium Xtreme Scanner is designed by Libelium to connect ZigBee, Wifi and Bluetooth sensors to the Internet through 3G connectivity (Meshlium Xtreme datasheet v. 4.3, 2014).



*Fig.39. The Libelium Meshlium Xtreme Scanner (Source: Libelium)*

The Meshlium Xtreme can function as:

- a ZigBee to Ethernet router for Wasmote nodes
- a ZigBee to 3G/GPRS router for Wasmote nodes
- a Wi-Fi Access Point
- a Wi-Fi Mesh node (dual band 2.4GHz-5GHz)
- a Wi-Fi to 3G/GPRS router
- a Bluetooth scanner and analyzer
- a GPS-3G/GPRS realtime tracker
- a Smartphone scanner (detects iPhone and Android devices)

In this project Meshlium Xtreme has been used as WiFi scanner in order to detect smarthphone devices. Each scanner has four antenn's. Only one antenna is needed for monitoring the Wi-Fi signal. Another antenna can be used for Bluetooth and the other two antenna's are for setting up the scanner as an access point, so the scanner can be used as a router. For this project just one antenna is needed, but actually three of them are used instead.

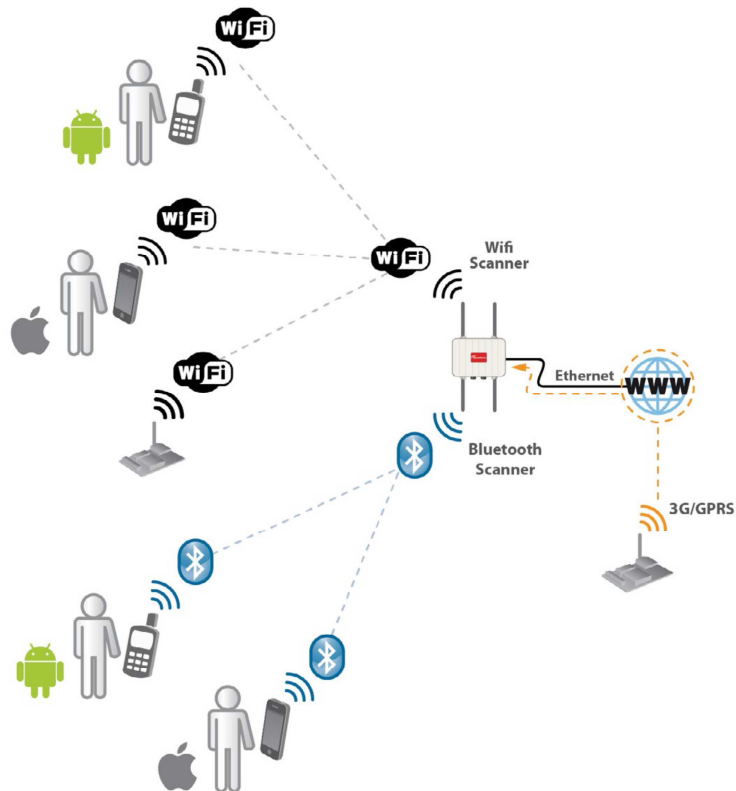


Fig.40. Conceptual schema of the working of a scanner (Source: Libelium)

The data received by the Meshlium always contains:

- The MAC address of the wireless interface, which allows unique identification
- The strength of the signal (RSSI), which may give an indication of distance to the scanning point.
- The vendor of the smartphone (Apple, Nokia, etc.)
- The TimeStamp, which indicates the date and time the data was collected

The collected data can be either stored locally on the Meshlium or stored in an external database.

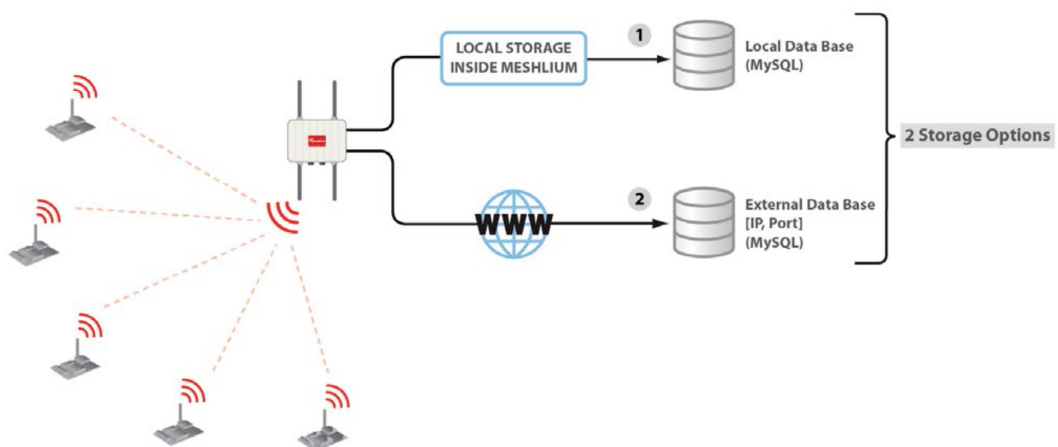


Fig.41. Ways to store the gathered data (Source: Libelium)

A Meshlium comes with the Manager System, an open source web application which allows to control quickly and easily the WiFi, ZigBee, Bluetooth and GPRS configurations along with the database storage options of the sensor data received (Meshlium Xtreme datasheet v. 4.3, 2014).

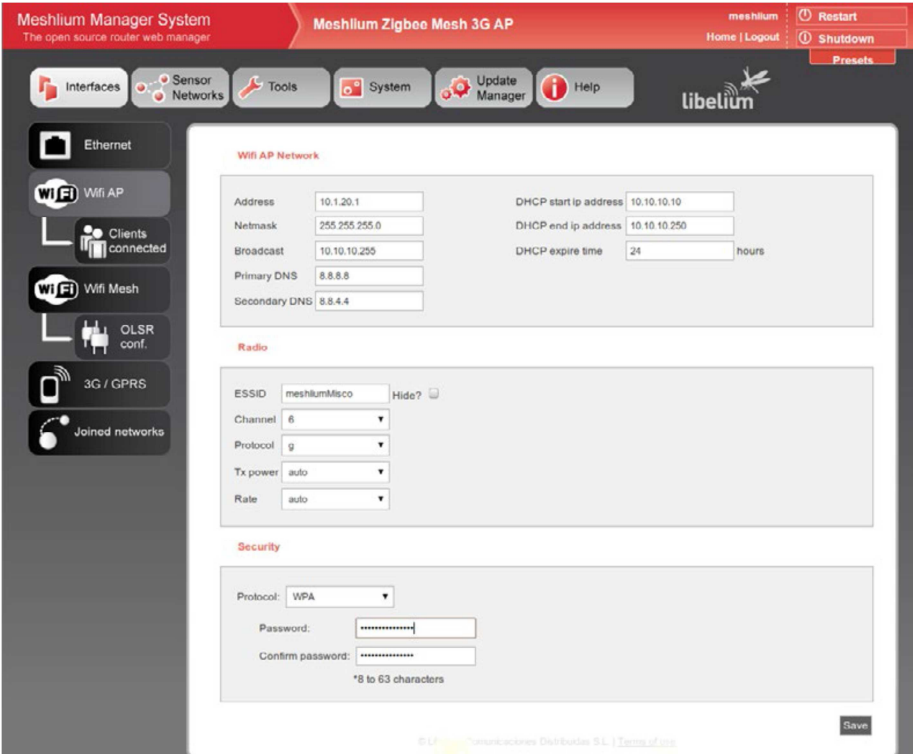


Fig.42. Manager System (source : Libelium)

7.2. TESTING WITH THE SCANNERS IN THE FACULTY OF ARCHITECTURE

The team performed several tests with the four Libelium Meshlium Xtreme scanners in the faculty of Architecture. The tests were held in a part of the West wing on the second floor (Figure 43).

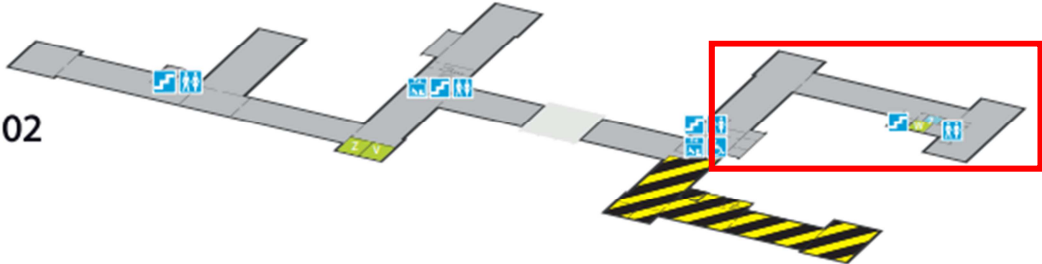
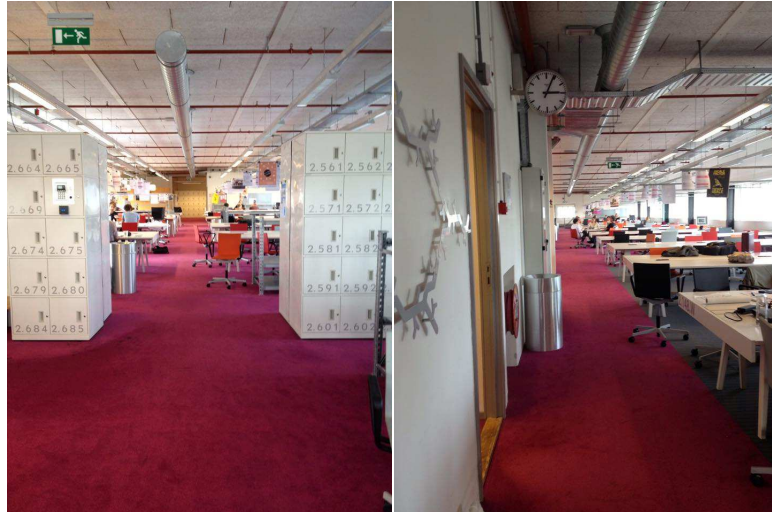


Fig. 43. Testing area in red

This environment was chosen because the space is similar to the environment of 'De Rotterdam'. The space consists of open spaces with free workspaces and small rooms made of glass and thin walls as seen in the figures below. This can be compared to the environment of 'De Rotterdam'.



*Fig. 44. Testing environment*

In this section of the second floor, the testing of different parameters on the Wi-Fi monitoring could be done, which resulted in 3 tests. The different parameters to find out by testing were:

- The range of scanning of a Wi-Fi monitor
- The influence of obstacles on the signal strength
- To distinguish areas based on signal strengths

For the testing procedure all available hardware was put into use. 4 Meshlium Xtreme (libelium©) devices were used and 2 Samsung Smartphones (see Figure 45).



*Fig.45. Materials used during the tests*

To keep all tests consistent, a few things were taken into consideration:

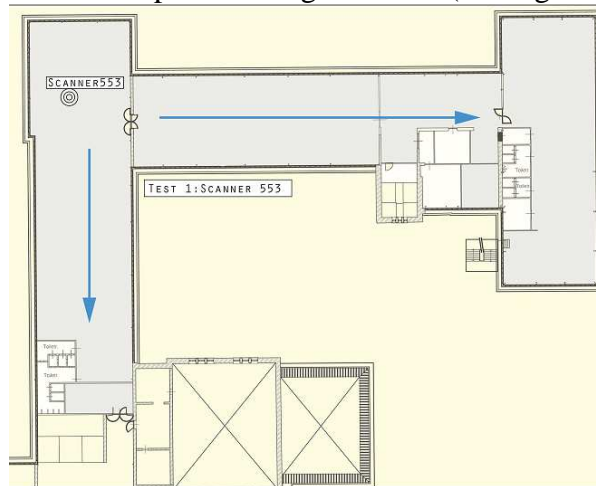
- The scanners and the phones were time synchronized
- The scanners were all set on a scanning time interval of 30 seconds
- The scanners were set to connect to an external database where all data was sent to
- The phones were in active mode during testing
- The phones were not connected to a network
- At least 2 phones were of the same brand and the same type
- Measuring was done in a straight line, at different points on the same distance from each other
- The team performed logged tests which provide the “ground truth” for the data. By analyzing this data in comparison to the logs of the tests, the localization precision and applicability can be tested.



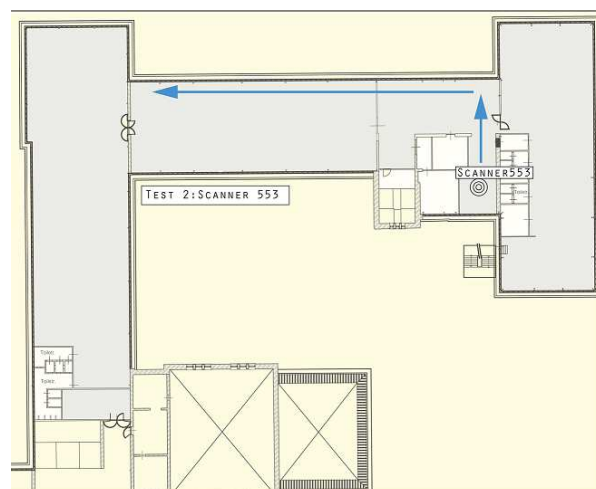


*Fig. 46. Test set-up*

For the first two tests, only one scanner was used. For test one the scanner was placed in open space, for test two the scanner was placed in a glass room (see Figure 47 and 48).

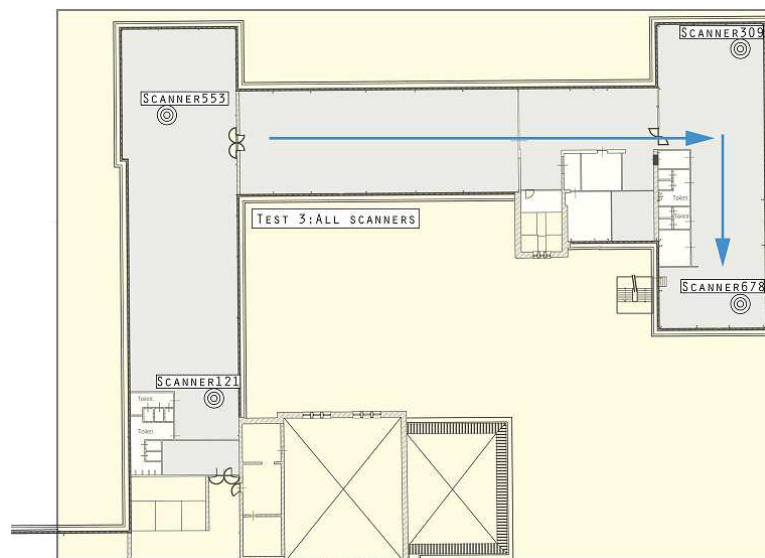


*Fig. 47. Scanner set-up test 1*



*Fig. 48. Scanner set-up test 2*

For the third test, all the scanners were used and placed only in the orange and green parts. Three of the scanner far away and at great angles from each other and one scanner was placed close to another, so as to explore the scanner interaction (see Figure 49).



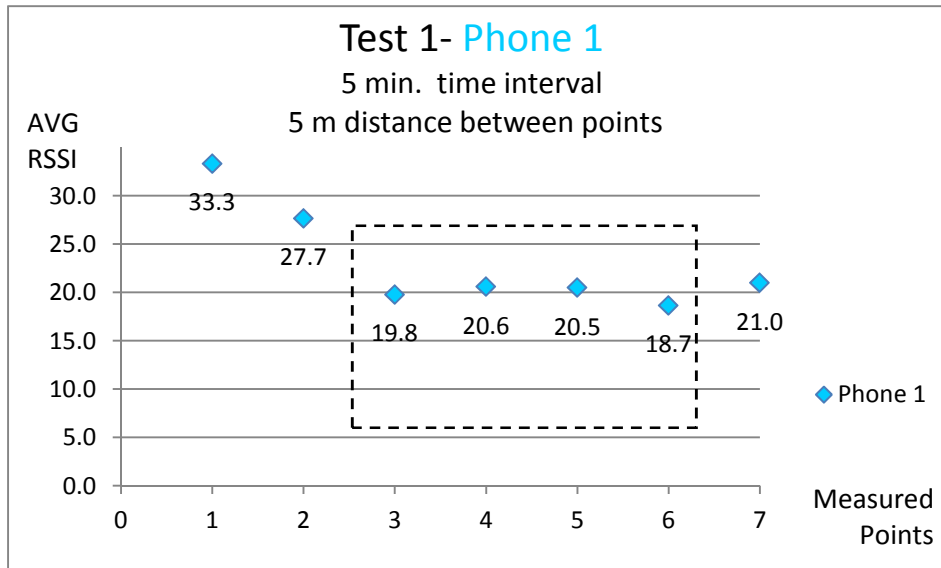
*Fig. 49. Scanner set-up test 3*

#### 7.2.1. THE INFLUENCE OF DIFFERENT BRANDS OF PHONES

First it was tested if all brands and types of phones were scanned by the scanner, by checking in the database how many times the team's own phones were scanned. What was observed was that iPhones were much less scanned than Android phones. A conclusion was made that iPhones were not suitable for testing (at least not the ones in this disposal), thus only Android phones were used. Since the client requested an Android application, this was not considered as a problem.

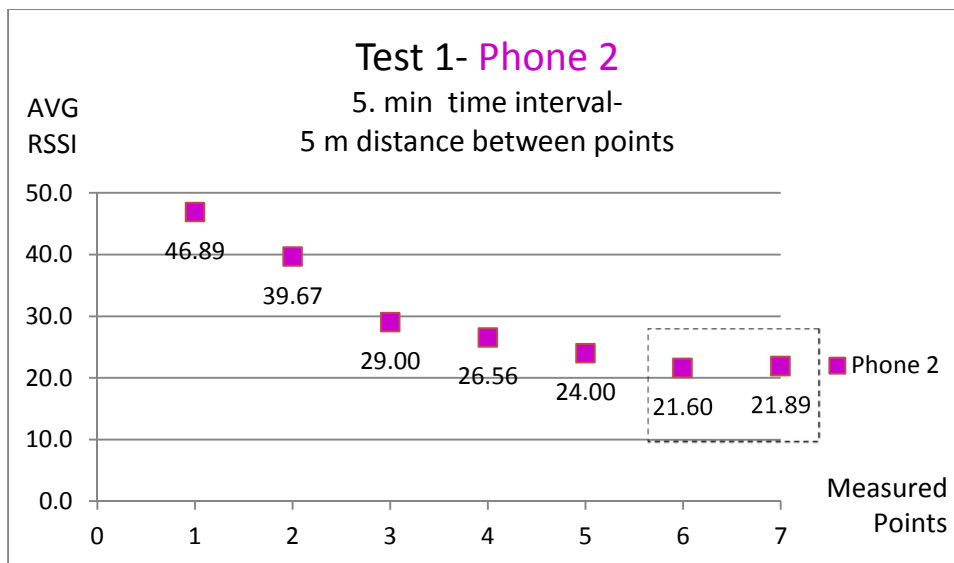
#### 7.2.2. TEST 1: RANGE OF SCANNING OF ONE WI-FI MONITOR

The first test was done to check the range of a scanner to see how the signal strength attenuates over distance in an open space. Although the received signal strength (RSSI) is dependent on many user-side factors such as if the phone is in stand-by mode, if the phone is in a pocket or held in hand etc., these factors were not taken into account as they lie with the user's usability on which the team cannot intervene. Thus, the tests performed assumed that the phone is placed on a table free of the human factor, in order to test how the RSSI decreased in the set environment. The scanner was placed in open space, the test was done in two different directions (blue and purple), faced away from the scanner (see the Appendix).



Graph 1. Result test 1

In the graph for the blue direction (Graph 1) can be seen that in open space (point 1 till 3), the RSSI decreases significantly, but when obstacles are present, such as rooms/ glass walls the decreasing of the signal gets less (see point 3 till 6). This can be due to multipath.

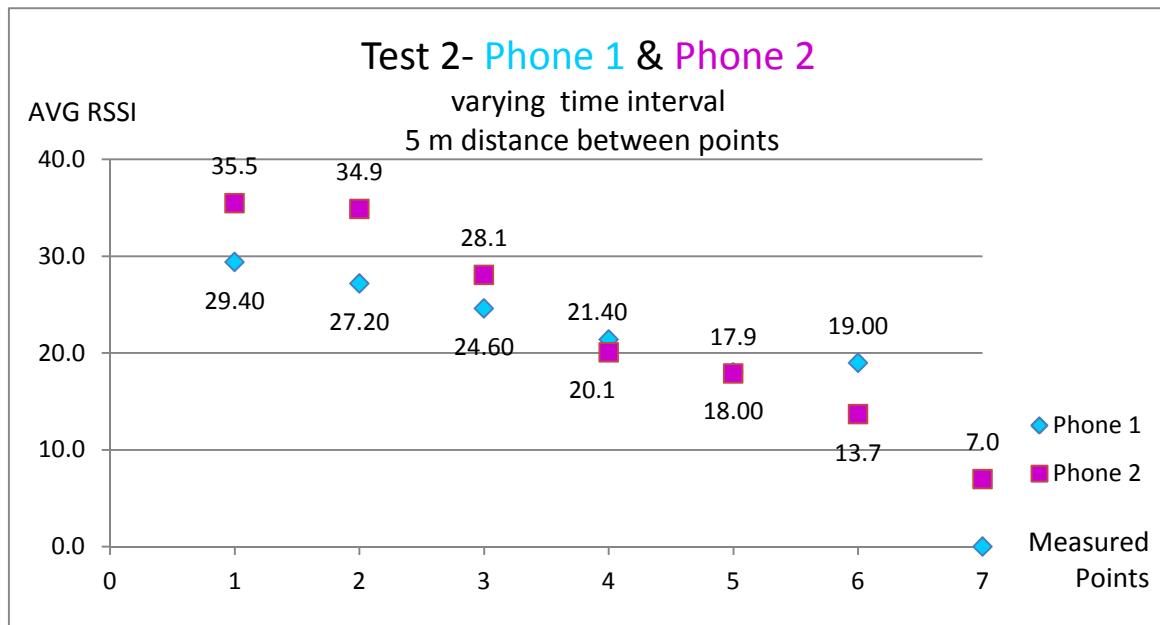


Graph 2. Result test 1

The purple path went mostly through open space. In Graph 2 it can be seen that the signal strength gets stable around point 6. Expected is that the signal strength get stable around a certain distance. To answer the test question, what is the range of the scanner, can be seen from the graphs that the range depends on the environment. To give some sort of indication from the test, the range is around 20 m, since every point is 5 m from each other and after 4 points the signal gets stable.

### 7.2.3. TEST 2: THE INFLUENCE OF OBJECTS

The second test was made with the purpose to empirically notice the influence of (thin) walls/ brick/ glass/ open or closed doors on the signal strength. This time a scanner was placed in a glass room.

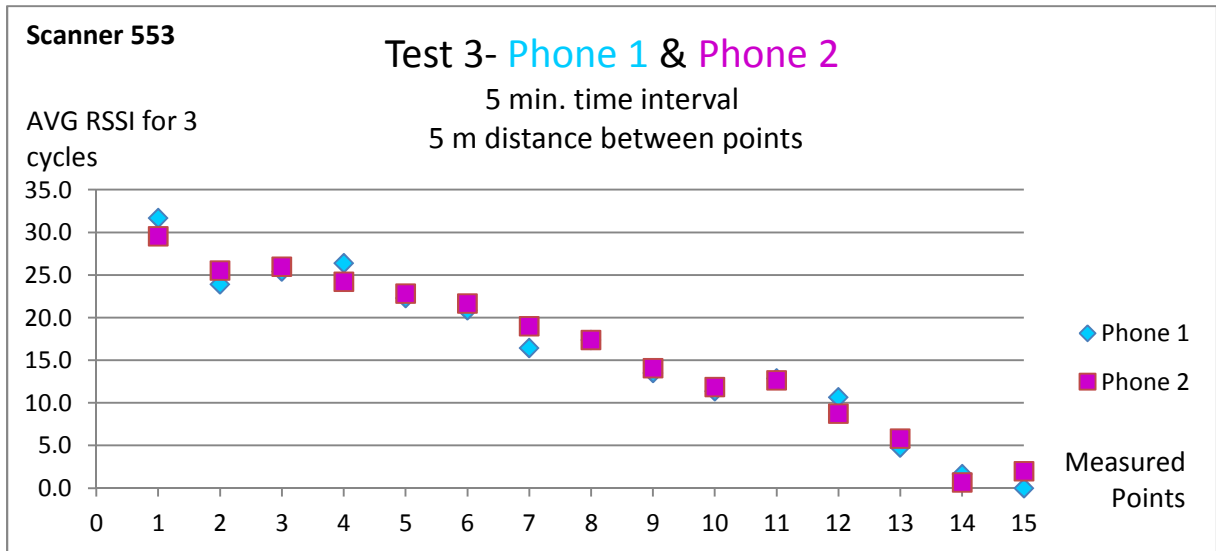


Graph 3. Result test 2

In the graph (Graph 3) it can be seen that, while using the same type and brand of smartphone, the RSSI values still fluctuate significantly. At some points the RSSI varies at most 7 dBm. This means that the signal strength as measured by the scanner for a smartphone is different, even though the measurements were at the same points. To answer the question, in comparison to the first test, can be seen that the influence of a material has a significant effect on the (fluctuation of) the signal strength. When there are a lot of obstacles around, the scanner detects a lot of variation between the signal strengths. When there is open space, phones fluctuate much less as visible in the next test.

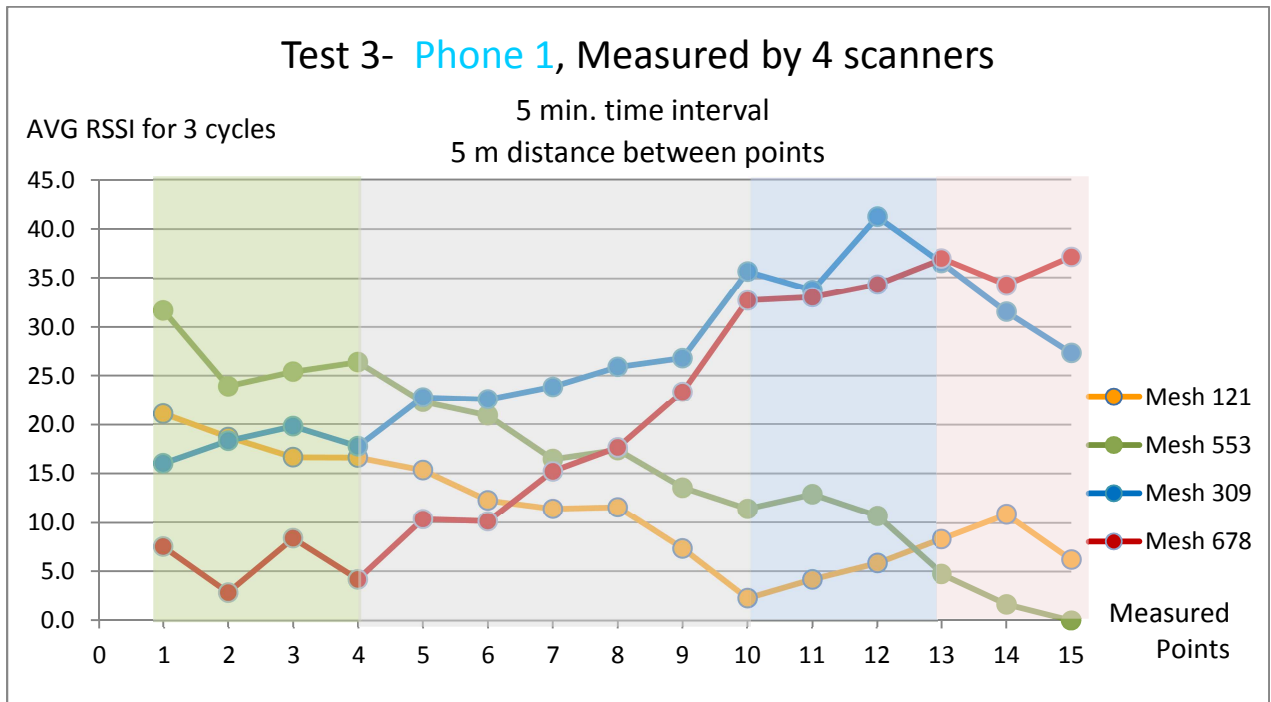
### 7.2.4. TEST 3: CAN AREAS BE DISTINGUISHED BASED ON SIGNAL STRENGTH?

For the third test, all the four scanners were placed in the testing environment, to see how well the phones were measured by each scanner and how the signals were interfering with each other. This to see if RSSI can be a viable measure in order to distinguish in which subdivision the scanned device belongs to. This test is of higher importance, because the results could be checked and an approximation of the accuracy of the localization can be conceived. The same test was done three times (three cycles).



Graph 4. Result test 3

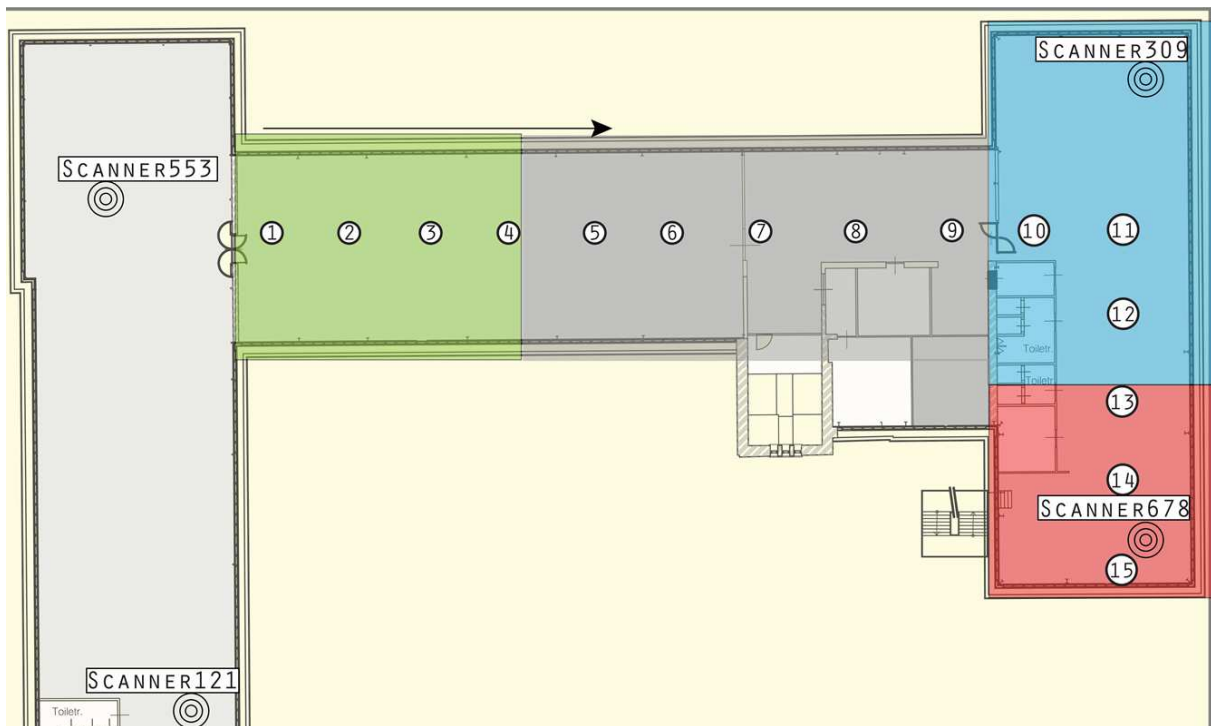
For each scanner, the AVG RSSI over all 3 cycles at all points is depicted in a result graph. The signal strength is decreasing, the further away from the scanner the points lie, as was expected. The graphs for the other three scanners portray the same results. In Graph 4 it appears that the fluctuation of the signals is not much, but that can be explained because the fluctuation is averaged out.



Graph 5. Result test 3

The graph above (Graph 6) gives an indication of how one phone is perceived by all the scanners (averaged over the three cycles) at every point. This graph compared to the floorplan gives an indication of the signal strength per subdivision (see Figure 50) and can be used for a very simple deterministic approach for determining in which subdivision a person could be.

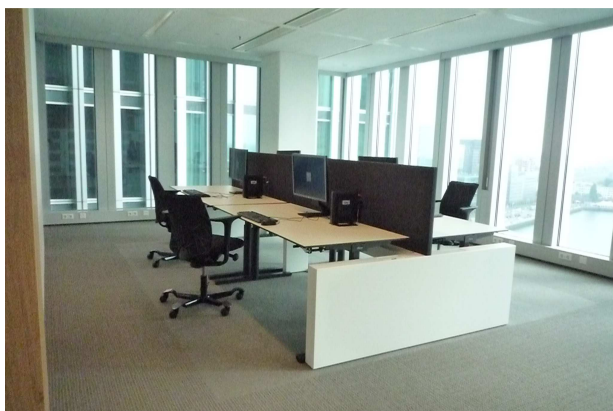
The distance between scanner309 and scanner 678 is probably a bit too small, that is why the lines in the graph are so close to each other. Placing the scanners further apart, will probably result in a more clear idea where a person could be located.



*Fig. 50. Space subdivision*

### 7.3. TESTING IN ‘DE ROTTERDAM’

After having tested the scanners in the faculty of Architecture, several tests were performed in ‘De Rotterdam’ building, in order to collect the data that will be used in the implementation of the ‘Catch-a-colleague’ application. The tests were held in the 16<sup>th</sup> floor of the building, where the environment consists mainly of open spaces with free workspaces and small rooms made of glass and thin walls (see pictures in Figure 51).





*Fig.51. Interior of the 16<sup>th</sup> floor of 'De Rotterdam' building*

In total 4 different tests were carried out with different scanner set-up.

In order to keep all tests consistent, a few things were taken into consideration:

- 4 Meshlium Xtreme and 3 Samsung Smartphones were utilized for the all four tests
- The scanners and the phones were time synchronized
- The scanners were all set on a scanning time interval of 30 seconds
- The data was collected in each point for a time interval of 5 minutes
- The data was stored locally on the Meshlium
- The phones were in active mode during testing

For the first 3 tests 22 testing points were measured, while for test 4 only 12 points were tested. The choice of the testing points was based on the environment, in order to test in areas with different characteristics: in the corridors (point 3 till point 14), in the open spaces (point 1, 15 and 16) and in the rooms (from point 17 till 22). The testing points were equally distributed among the whole floorplan and placed at a distance fairly constant from each other, especially for the points along the corridors. Each test will be described in detail in the following paragraphs.

### 7.3.1 TEST 1

In the first test the scanners were placed in open space at the corners of the building, as far away as possible and at great angles from each other in order to explore the scanners interaction (see Figure 52 and 53).



*Fig.52. Scanner placement test 1*

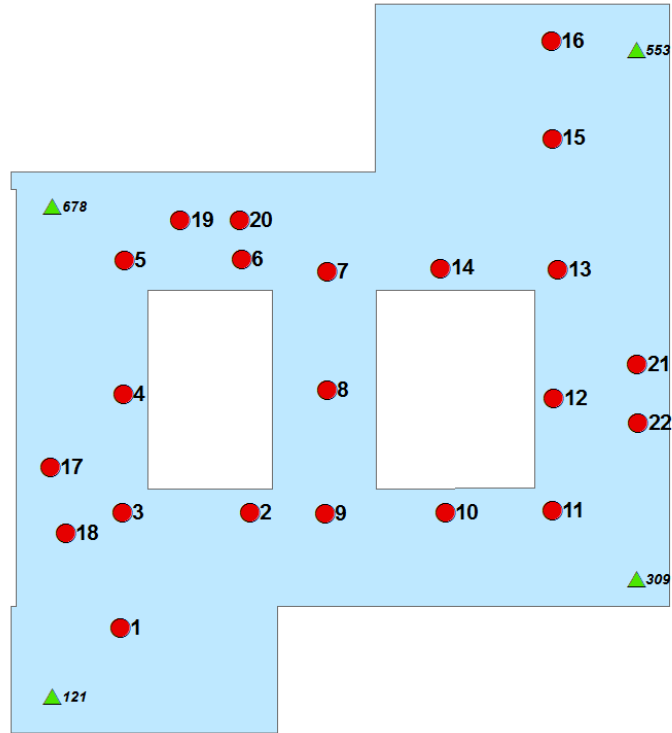
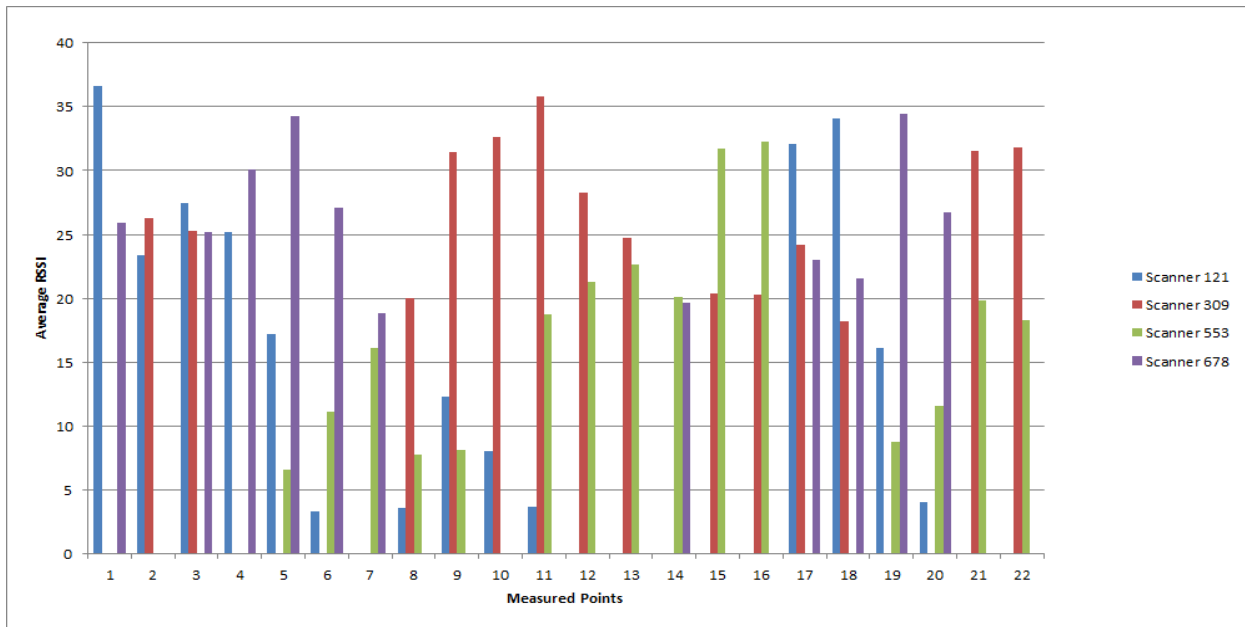


Fig.53. Set-up test 1

This test was performed twice and in the end all the data collected was combined and the average RSSI was computed. The results are shown in the graph below.

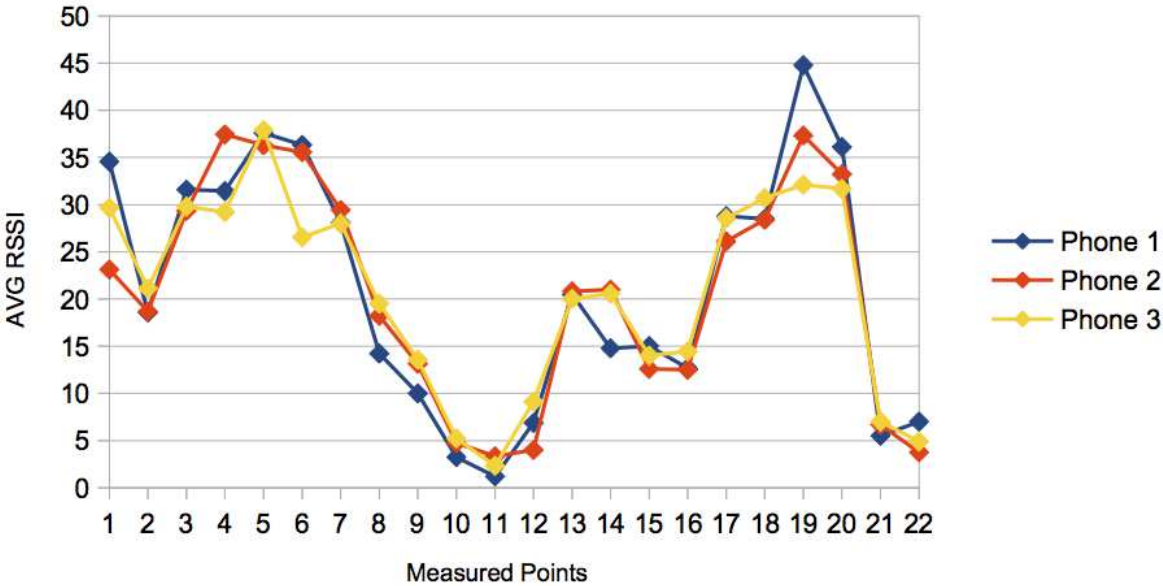


Graph 6. Results test 1



The graph gives an indication on how the three phones are perceived by all the scanners (averaged over the two cycles) at every point. In general, the values measured with this scanner set-up may give an indication of the signal strength per subdivision, in which a person could be located. However, in some points (1, 4, 7, 10, 12, 14, 15, 16, 21 and 22) only two scanners were visible and for this reason some localization methods like trilateration can hardly be implemented with this scanner layout, since at least three scanner needs to be visible. For some points, the phones were scanned only few times within the time interval and therefore the AVG RSSI measured should not be considered as reliable. For this reason, the AVG RSSI values of the points that were scanned less than 10 times in the 5 minutes interval were considered to be zero.

Moreover, it may also be interesting to see the fluctuation between the values according to the different three phones, which are of the same type and brand. In the graph below the AVG RSSI values for scanner 678 for each phone are represented.



Graph 7. AVG RSSI values for scanner 678

7.3.2. TEST 2

For the second test the scanners were placed much closer to each other, at the corners of the central concrete block. The same 22 points that were used in test 1 were measured here (see Figure 54 and 55).



Fig.54. Scanner placement test 2

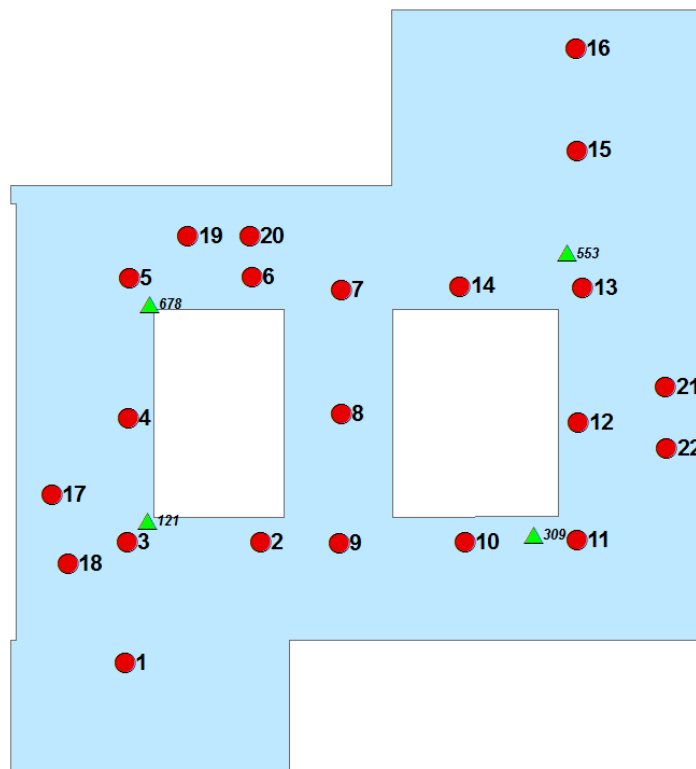
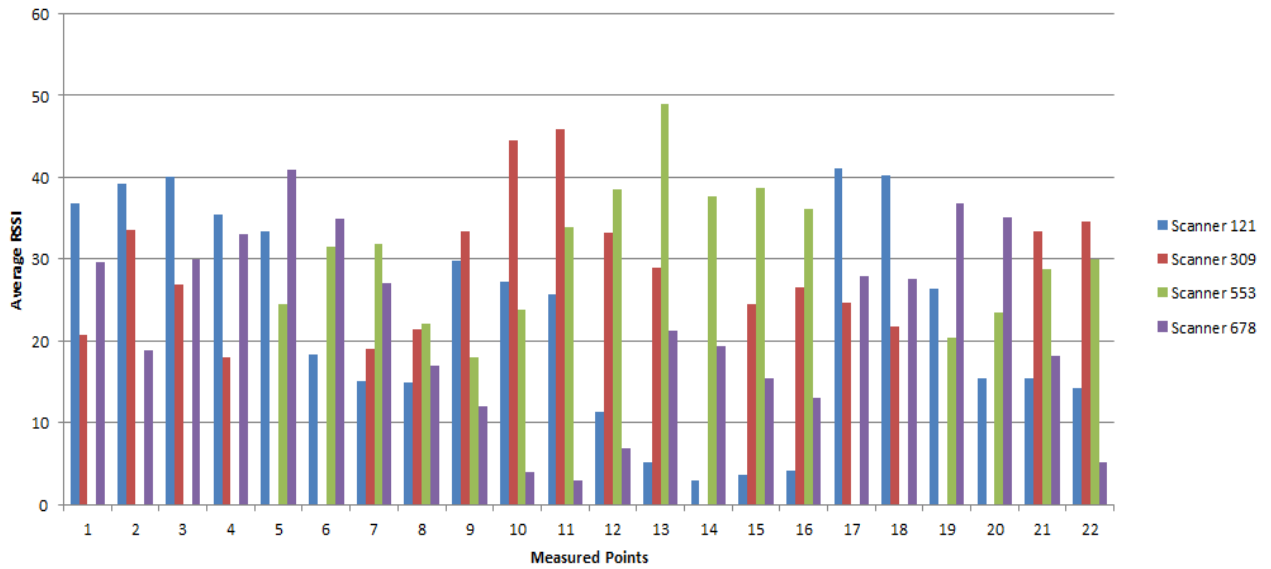


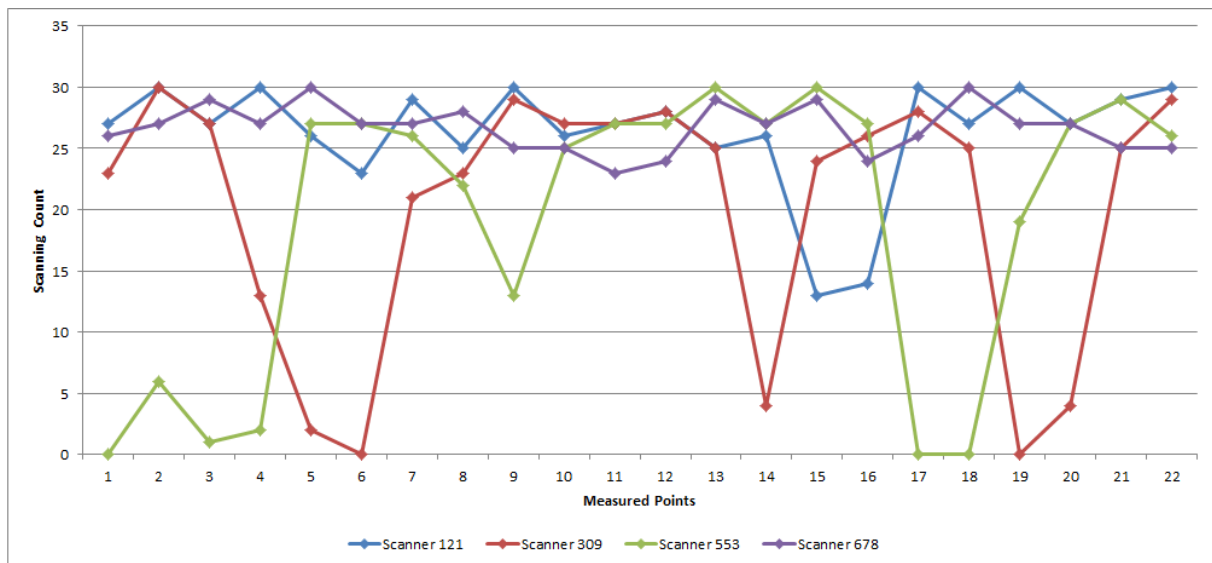
Fig.55. Set-up test 2

Also for this test, two cycles were performed. The average result of the two cycles can be seen in the following graph which shows the AVG RSSI measured at each point for the four different scanners. It is important to notice that in this case for each point at least three scanners were always visible and this means that this scanner layout may be more suitable for localization methods like trilateration.



Graph 8. Results test 2

Another aspect to take into consideration is the number of times the phones are scanned at each point. Since the scanning rate is 30 seconds and the data is collected in each point for a time interval of 5 minutes, in total for all the three phones there should be 30 scans (10 for each phones). However, from the graph below it can be seen that not for all the 22 points the scanning count is 30; for some points the number of scans is rather low for a specific scanner.



Graph 9. The number of times a scanner has seen the three phones during test 2

### 7.3.3 TEST 3

In test 3 each of the scanners was placed on one side of the central concrete block and two of the scanners were placed inside two rooms (scanner 121 and scanner 553) in order to

empirically notice the influence of walls on the signal strength. Also in this case the same 22 points were measured but the test was only performed once (see Figure 56 and 57).



Fig.56. Scanner placement test 3

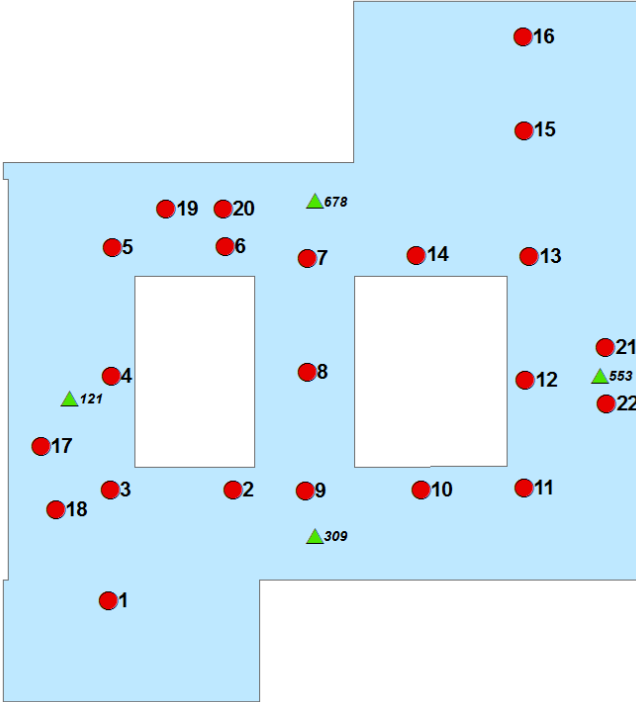
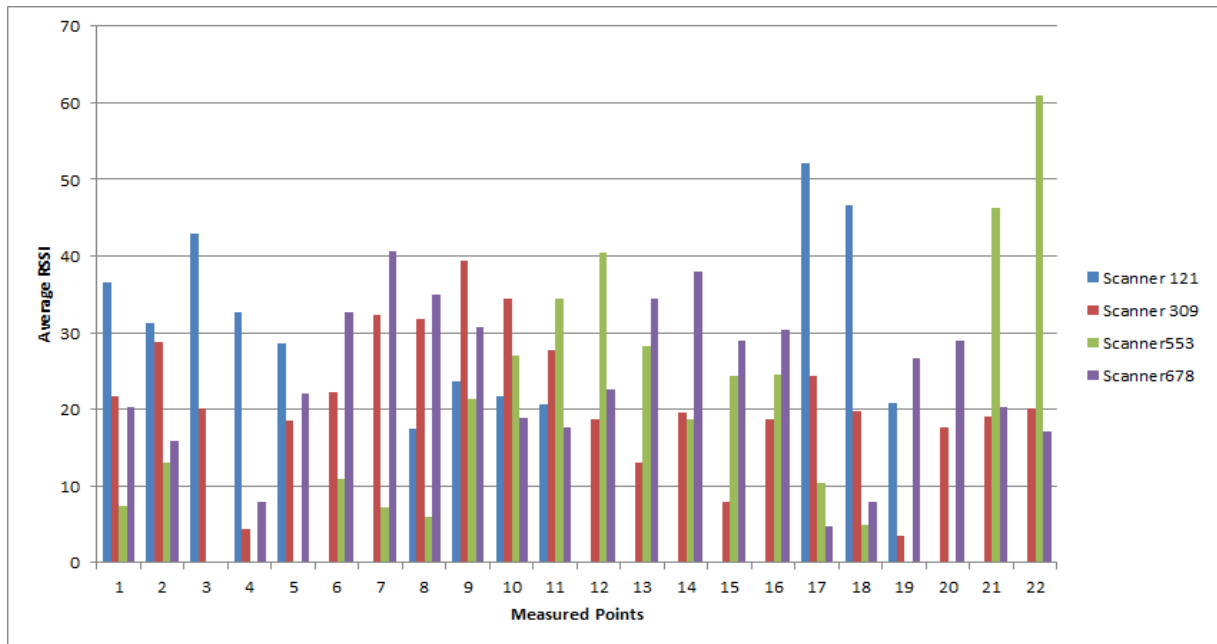


Fig.57. Set-up test 3

The result of the test are summarized in the graph below, which shows the influence of walls in attenuating the signal strength. For instance, point 21 and 22 are at the same distance from the scanner 553, but point 21 has a lower AVG RSSI value than point 22 probably because of the presence of a wall between it and the scanner. Moreover, it is important to notice that scanner 121 and 553 that were placed in the rooms are not visible from a lot of points and this may be related to the signal attenuation due to the presence of walls and obstacles.



Graph 10. Results test 3

### 7.3.4 TEST 4

In test 4 only half of the floor plan was considered and the four scanners were placed and closer to each other. This set up was used to test if with more scanners could possible result in a higher accuracy. For the project only four scanners were used, but potentially more scanners could be placed in one floor. In this case the RSSI was measured only in 12 points, which were also used in the previous tests (see Figure 58 and 59).



Fig.58. Scanner placement test 4

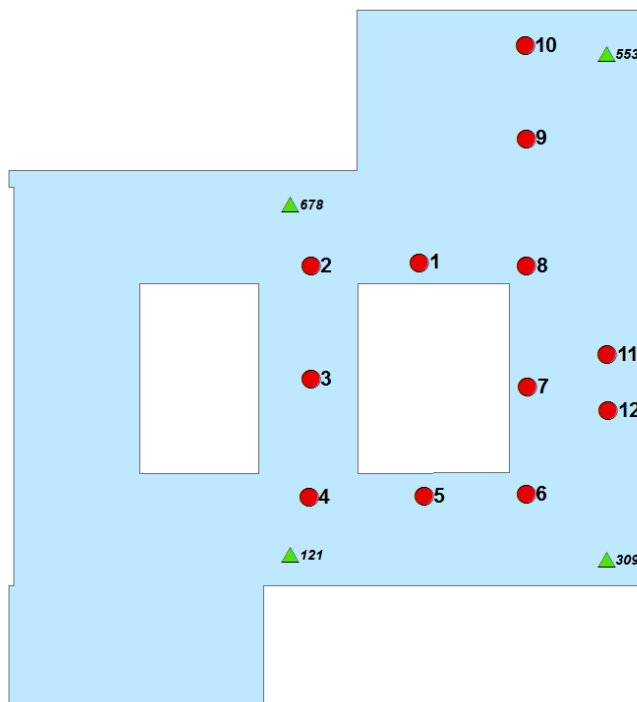
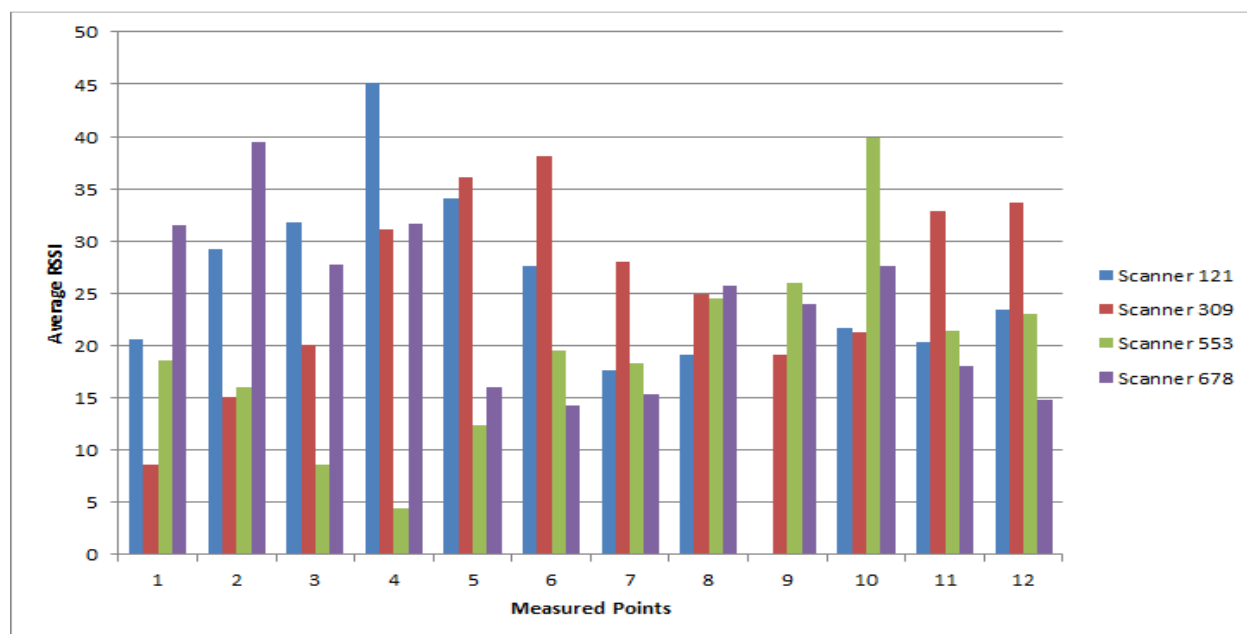


Fig.59. Set-up test 4

The following graph shows the result of the test. In general, it can be seen that for almost all the points all the four scanners are visible. However, for some points pretty similar values were measured and this may be a problem for the correct localization of a person in an area rather than another.



Graph 11. Results test 4

## 8. System Engineering

This chapter describes all the used software and tools.

### 8.1. SOFTWARE USED

For all the components whereof the prototype exists, different software and tools have been utilized. However, almost for all components, the Python programming language has been utilized, as well as a GIS software and the database MySQL workbench. In the table below, all the tools that have been used for each component are summarized.

Components	Tools
Space Subdivision	<b>Intuitive Subdivision</b> <ul style="list-style-type: none"> <li>• AutoCAD software</li> <li>• GIS software</li> </ul>
	<b>Automatic Subdivision</b> <ul style="list-style-type: none"> <li>• GIS software</li> <li>• Triangle Shewchuk software (Constrained Delaunay Triangulation)</li> <li>• Python libraries PyShape and Shapely</li> </ul>
Localization	<b>Multi-iteration</b> <ul style="list-style-type: none"> <li>• GIS software</li> <li>• Python libraries MySQLdb, Fiona, Shapely</li> <li>• MySQL workbench</li> </ul>
	<b>Fingerprinting</b> <ul style="list-style-type: none"> <li>• GIS software</li> <li>• Python library Scipy (scipy.interpolate.Rbf method)</li> <li>• MySQL workbench</li> </ul>
Navigation	<b>Manual Network</b> <ul style="list-style-type: none"> <li>• GIS software</li> </ul>
	<b>Semi-Automatic &amp; Automatic Network</b> <ul style="list-style-type: none"> <li>• GIS software</li> <li>• Python library Pyshape and Shapely</li> </ul>
	<b>Path-finding</b> <ul style="list-style-type: none"> <li>• Python libraries (Dijkstra algorithm)</li> </ul>
Visualization	<b>Application</b> <ul style="list-style-type: none"> <li>• Eclipse</li> <li>• MySQL workbench</li> <li>• PHP</li> <li>• Unity3D</li> </ul>

*Table 4: Components and tools used to create the prototype*

The following chapters describe the implementation of the different components and the integration in a technical manner. The decision making and analysis are described as well as the end results.

## 9. Space subdivision

An important component of the ‘Catch-a-Colleague’ application is the space subdivision of indoor space, which is fundamental for correctly guiding an employee to the location of another colleague but also for testing the localization algorithm. Since the indoor environment can be very complex, different aspects have to be taken into account while modelling and subdividing indoor space, such as obstacles like furniture, columns and walls.

### 9. 1. IMPLEMENTATION

In order to subdivide the space in ‘De Rotterdam’ building, two different implementations were carried out by the team: the intuitive space subdivision and the automatic space subdivision, which is based on the Multi- Layered Space Model from IndoorGML. Each of them will be explained in detail in the next paragraphs.

For the distribution of the space a few requirements were set-up by the team:

The subdivisions should:

- all have the same size
- consists of around eight subdivisions (based on the localization accuracy)
- Not distinguish between rooms/ open space. A subdivision can consist of either open space, rooms or both
- The same subdivision for each floor (the floors all have approximately the same layout, thus the same subdivision can be used for all floors)

#### 9.1.1 Intuitive space subdivision

The intuitive space subdivision is based on subdividing the space in a human-understandable way, considering the characteristics of the building (obstacles, rooms, etc.), visibility criteria (e.g. line of sight) and the usage of space (workspaces). Although this solution might lead to a better human-understandable result, it is quite hard to be implemented in an automatic way, since the environment must be modelled accurately as well, which is time-consuming.

The subdivision was made from different regions in CAD software and then converting them into the Shapefile format using GIS software. Before performing this operation, the original floor plan provided by the Municipality of Rotterdam has been changed: the geometry was simplified and semantics information was attached to the geometric components. Each entity has an ID and information about the geometric data type (e.g. polyline), the type of object (e.g. inner\_walls, interior, inner\_facade, concrete\_walls) and the color (see figure 60).

The two holes that can be seen in figure 61, consists of the concrete core where the elevators and stairs are. Since the localization is not that accurate too be localized in the concrete core, holes are made of these two subspaces.



Table								
Export_Output.txt								
FID	FID	Entity	Layer	Color	Linetype	Elevation	LineWt	Reflame
289	0	LWPolyline	14_ConcreteWalls	3	Continuous	0	25	
324	0	LWPolyline	14_InnerFacade	2	Continuous	0	25	
359	0	LWPolyline	14_InnerWalls	1	Continuous	0	25	
409	0	LWPolyline	14_ConcreteWalls	3	Continuous	0	25	
412	0	LWPolyline	14_ConcreteWalls	3	Continuous	0	25	
415	0	LWPolyline	14_ConcreteWalls	3	Continuous	0	25	
515	0	LWPolyline	14_Interior	8	Continuous	0	25	
542	0	LWPolyline	14_Interior	8	Continuous	0	25	
547	0	LWPolyline	14_Interior	8	Continuous	0	25	
552	0	LWPolyline	Outline	6	Continuous	0	25	
553	0	LWPolyline	Outline	6	Continuous	0	25	
554	0	LWPolyline	Outline	6	Continuous	0	25	
555	0	LWPolyline	Outline	6	Continuous	0	25	
556	0	LWPolyline	Outline	6	Continuous	0	25	
557	0	LWPolyline	Outline	6	Continuous	0	25	
558	0	LWPolyline	Outline	6	Continuous	0	25	
559	0	LWPolyline	Outline	6	Continuous	0	25	
560	0	LWPolyline	Outline	6	Continuous	0	25	

Fig.60. Attribute table `De Rotterdam` floor plan with semantics information.

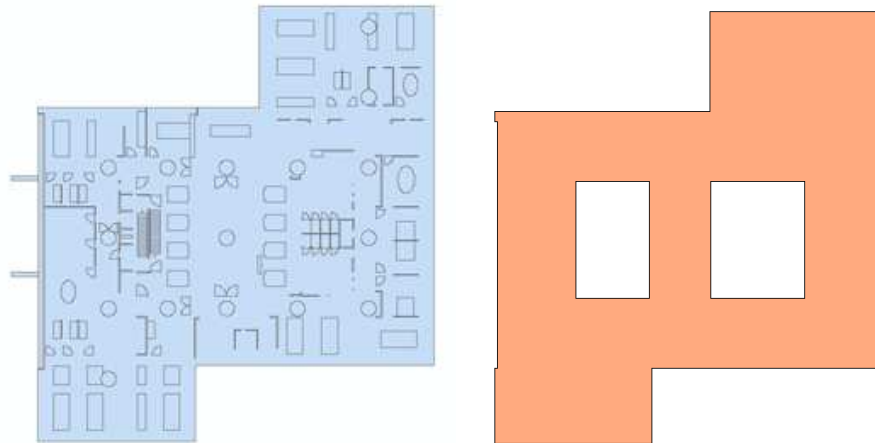
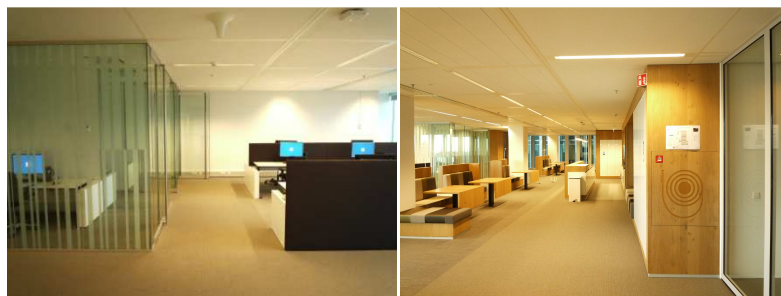


Fig. 61. Floor plan simplification.

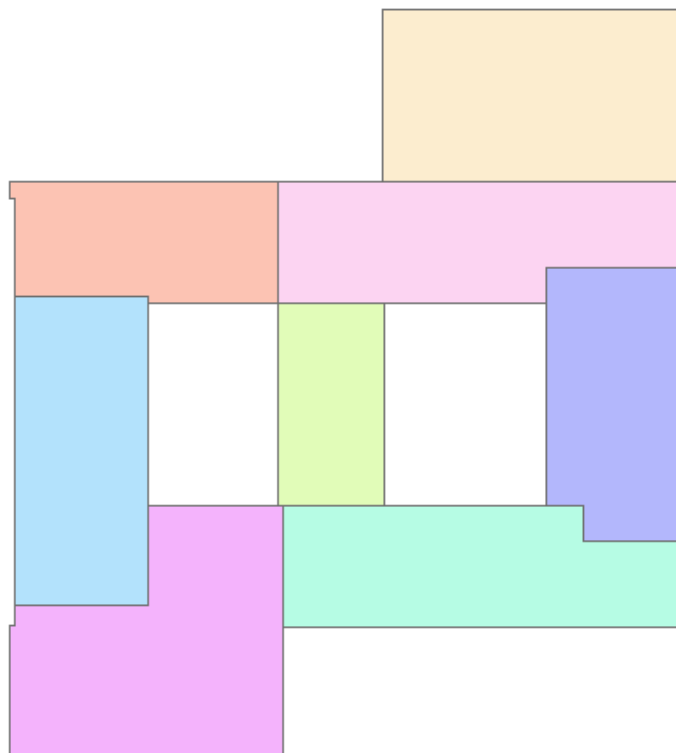
The intuitive space subdivision is based on the different workspaces that can be distinguished in the floor plan. If you look at the floor plan, the space per floor seems to be very big and maybe confusing. But when walking there, the space seems smaller and it already seems to be very logically divided into several workspaces (see pictures in Figure 62).





*Fig. 62. Several workspaces on the 16<sup>th</sup> floor*

These workspaces were the input for the intuitive space subdivision. In the end eight different subspaces were created (see Figure 63 below).



*Fig. 63. Intuitive space subdivision.*

An advantage of the intuitive space subdivision is that it is easy to understand and looks very logical. However, the operation of drawing a space subdivision manually might be time consuming, especially if the space has to be manually subdivided for all the floors (which is not the case for ‘De Rotterdam’ building, since the floors are about the same). Another disadvantage is that by manually dividing the space, there lies a possibility that invalid polygons are made. This can give difficulties in the next steps for deriving the network. An automatic way of subdividing the space has therefore also been tested and will be described in the next paragraph.

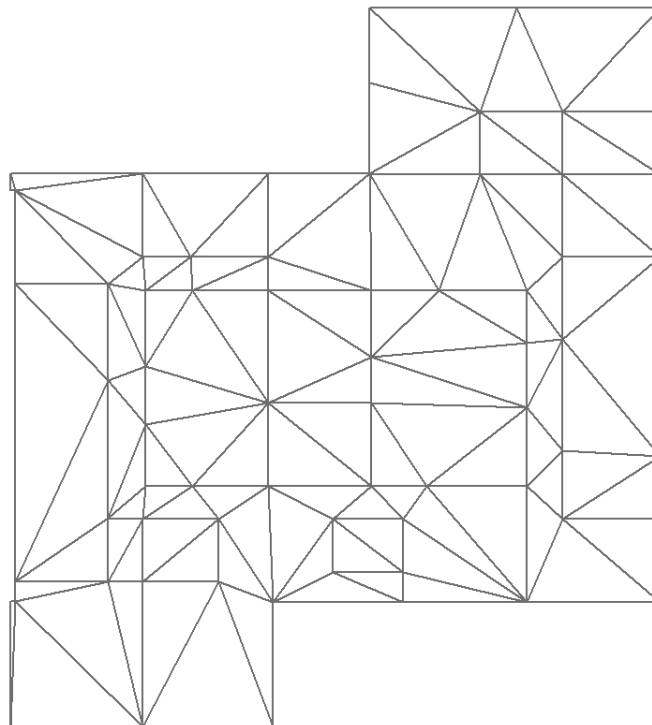
### 9.1.2 Automatic space subdivision

In order to automatically subdivide the floor plan of `De Rotterdam` building, the open space needs to be divided. This can be done by performing a triangulation on the open space, or dividing the space in a grid or using the Voronoi Diagram for example. Triangulation was chosen, because the team has experience with performing a triangulation.

The Constrained Delaunay Triangulation has been carried out to decompose the polygon of the floor plan into triangles forming a planar partition. The Constrained Delaunay Triangulation is one of the most suitable methods for surface approximation because it has several advantages over other triangulation methods:

- The triangles are as equi-angular as possible and skinny triangles are avoided
- Ensures that any point on the surface is as close as possible to a node
- The triangulation is independent of the order the points are processed

To perform the Constrained Delaunay Triangulation, the Triangle software package, which computes high-quality unstructured triangular meshes, was utilized. In addition, Pyshape and Shapely, Python libraries were used to script the algorithm. The figure below shows the result of the Constrained Delaunay Triangulation on the floorspace.



*Fig.64. Constrained Delaunay Triangulation.*

The triangles created by the triangulation were then combined with the range of the Meshliums, derived by the heat maps generated with the fingerprinting localization method (more in chapter 8). The reason to combine the triangles with the range of the scanners would result in subdivisions that fit the accuracy of the localization method and could create the 'best' subdivision to create the highest accuracy of the localization method.

### 9.1.3 Combination of the triangulation with the heat maps

The triangulation was combined with the heat maps, generated by interpolation in the fingerprinting localization method (more in section 8.3) (see Figure 65).

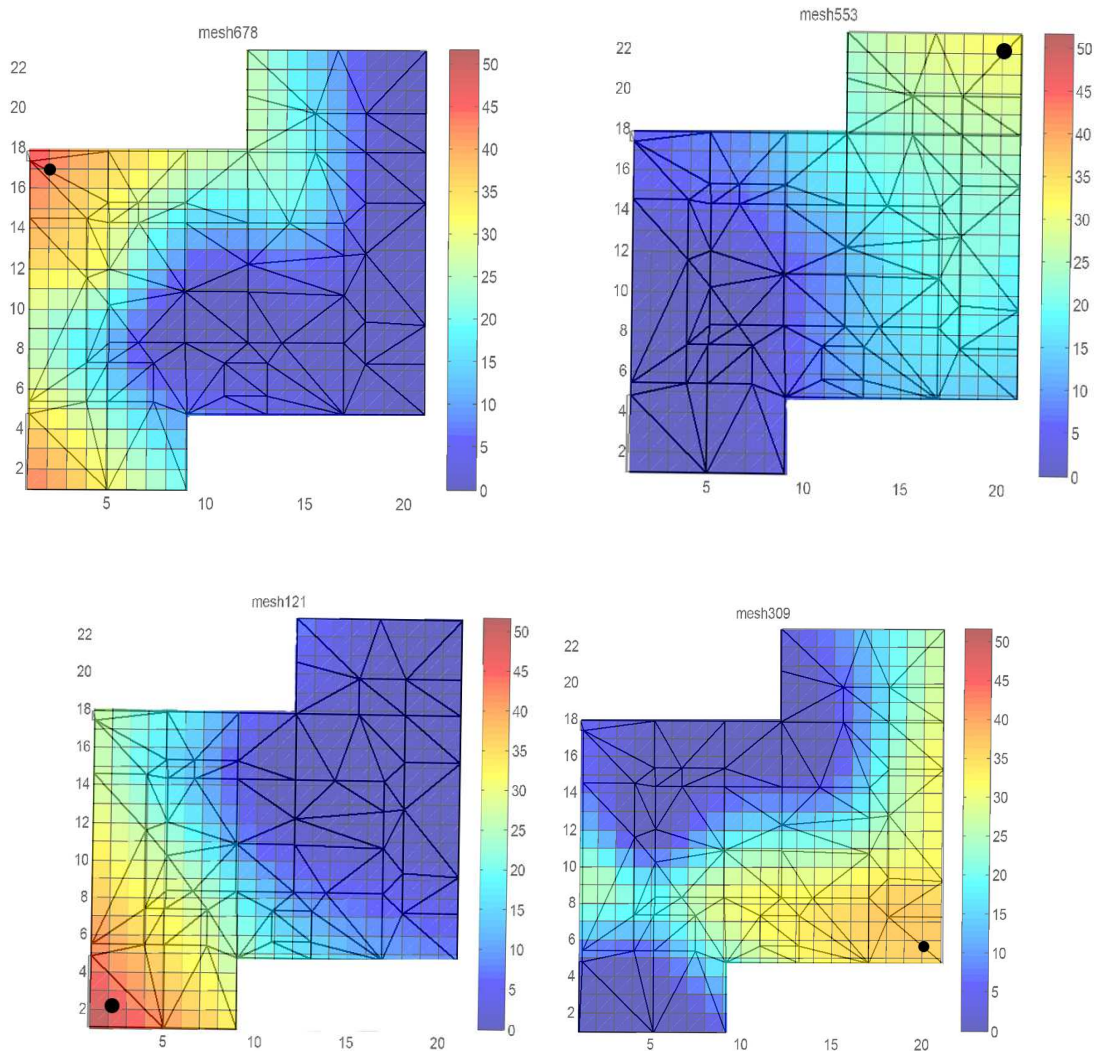
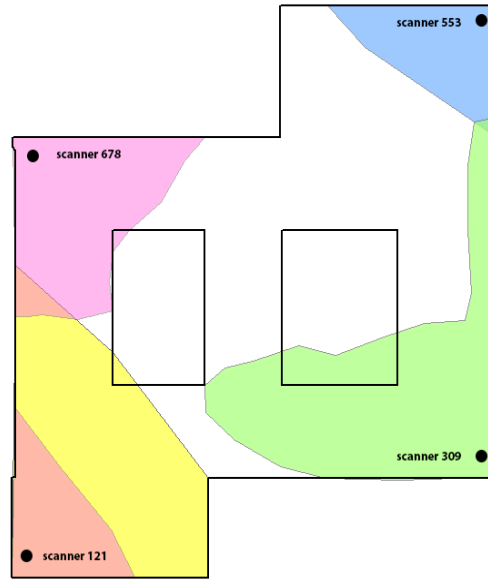


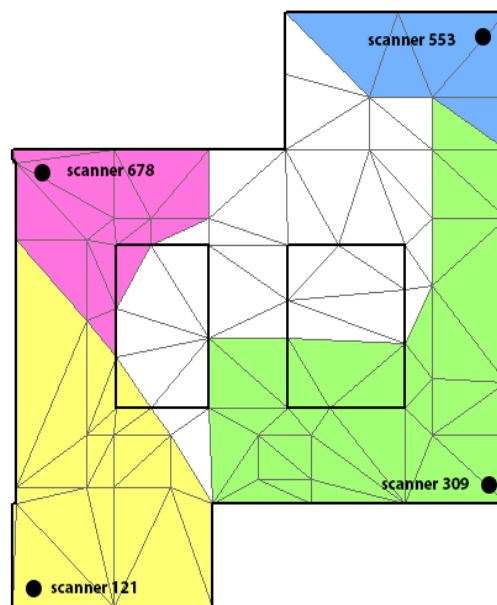
Fig.65. Combination of the triangulation with the heat maps for scanner layout 1.

In this case RSSI values not smaller than 30 db were considered and an irregular area was created around each scanner with the cells (from red to yellow) that fell into this range. RSSI values smaller than 30 db were not taken into account since they were not considered as reliable. As it is shown in the figure below, some areas overlap and especially for scanner 678 an anomalous fact has occurred since high values are perceived near scanner 121.



*Fig.66. Irregular buffers according to RSSI values for scanner layout 1.*

These areas were combined with the triangles generated by the Constrained Delaunay Triangulation. In this way all the triangles that fell into one of scanner areas were considered to be part of the same subspace and they were colored accordingly. For the case in which a triangle fell into more than one area, the heat maps were consulted and the triangle was assigned to the scanner that had the highest RSSI value for that triangle. In the end four different subspaces were obtained, plus one big white subspace in the middle (see Figure 67 below).



*Fig. 67. Space subdivision combining triangulation with heat maps for scanner layout 1.*

The same procedure has been carried out for scanner layout 2, in which the scanners are placed closer to each other (see Figure 68).

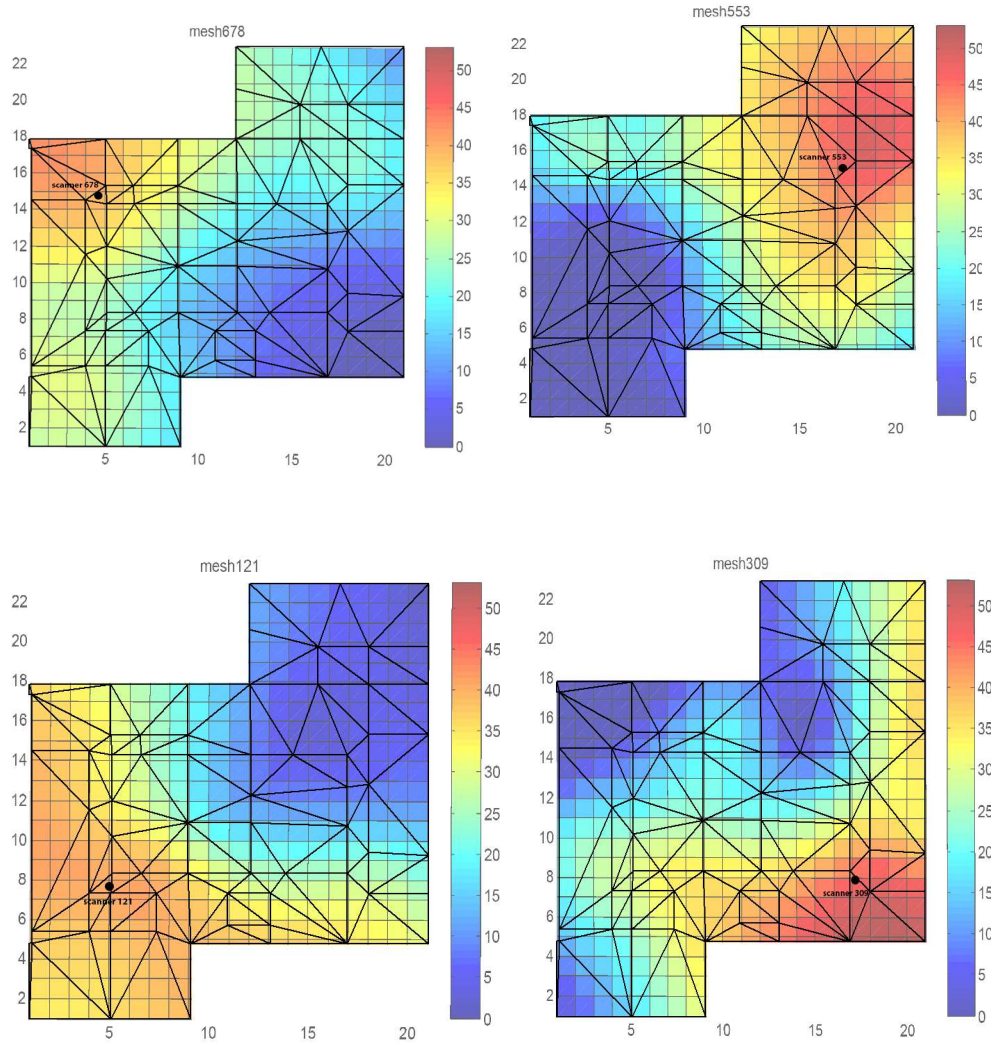


Fig. 68. Combination of the triangulation with the heat maps for scanner layout 2.

In this case the range of the scanners overlaps more often (see Figure 69 below).

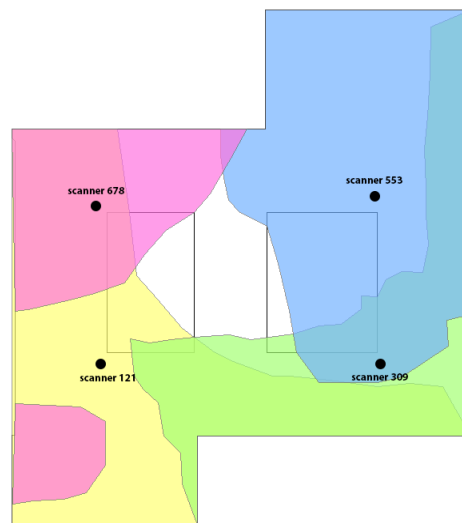


Fig. 69. Irregular buffers according to RSSI values for scanner layout 2.

The end result looks like the Figure 70 below, the floor plan is split into just four subspaces, except for two triangles that didn't fall into any buffer.

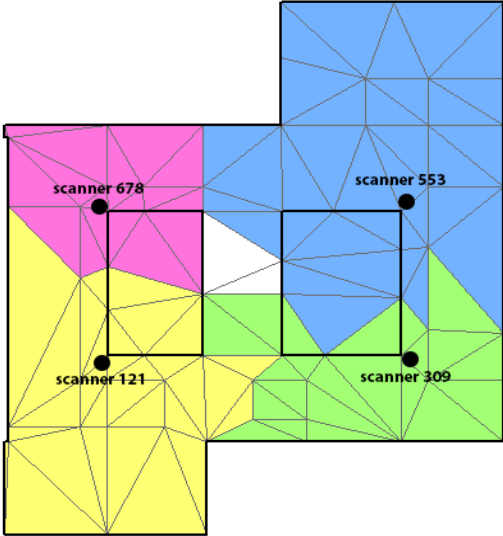


Fig. 70. Space subdivision combining triangulation with heat maps for scanner layout 2.

The combination between the triangulation and the heat maps has been done using GIS software. Moreover, since with this approach just four or five subspaces are created, it leads to a rather too coarse subdivision that it is not suitable for the localization. For this reason, this approach has not been tested and automatically implemented in code. A similar approach has been implemented and will be described in the next section.

9.1.4 Combination of the triangles with a radial buffer concerning the range

This method was implemented first for scanner layout 1. Buffers with a radius of 10 meters were built around each scanner and all the triangles that fell into the circle were considered as part of one subspace. In this way four different subspaces were created around the scanners (see Figure 71 below).

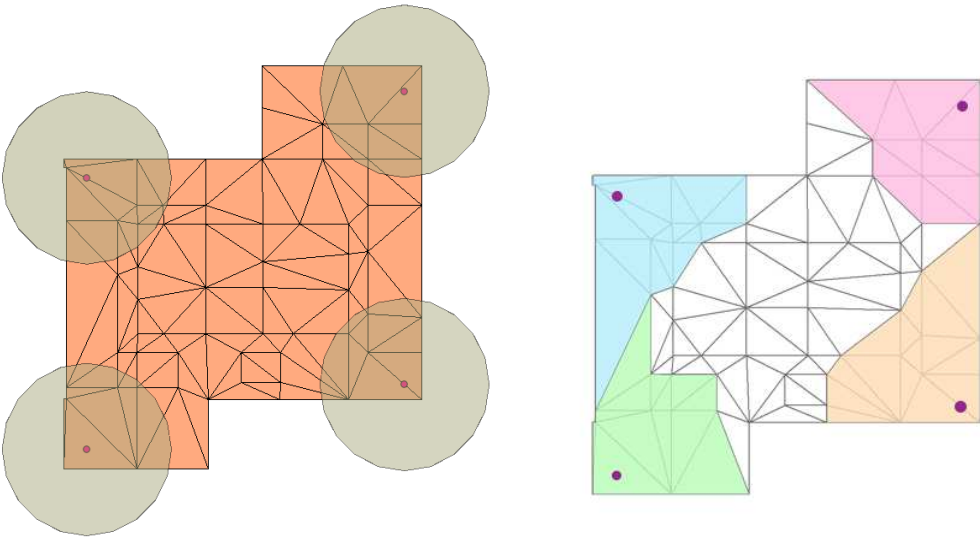
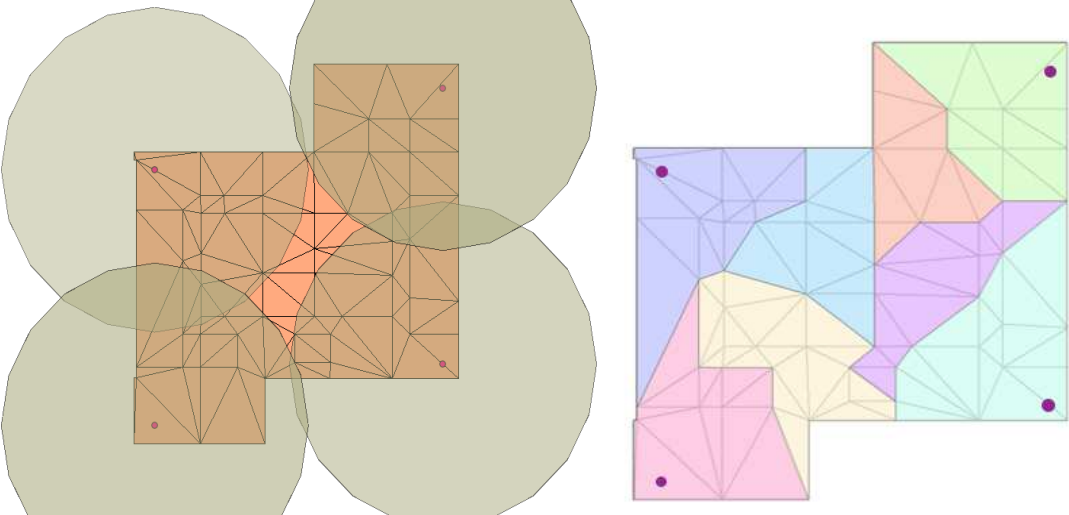


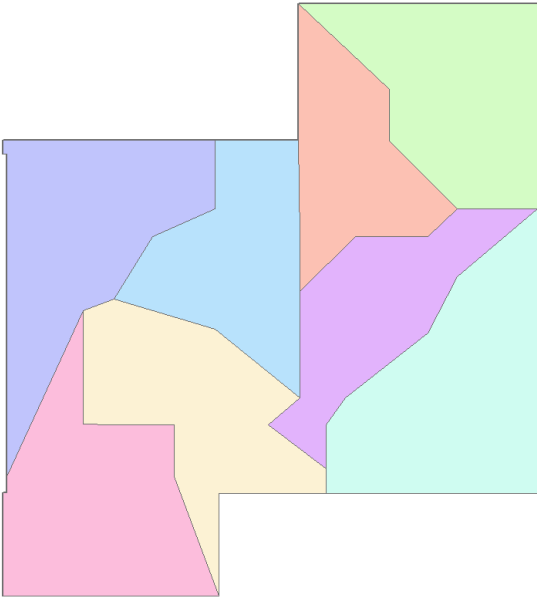
Fig.71. Range of the scanners 10 meters (left) and 4 subspaces (right).

The white space in the middle of the floor plan (Figure right) was not covered by the 10-meters range of the scanner and therefore it still needed to be subdivided. For this reason, another buffer of 20 meters radius was carried out around the Meshliums and 4 other subspaces were generated (see Figure 72 below).



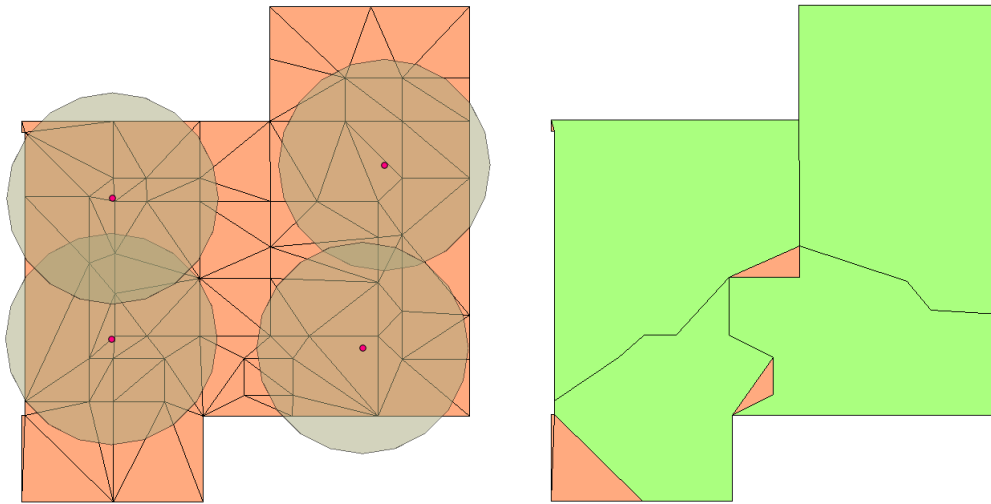
*Fig.72. Range of scanner 20 meters (left) and different subspaces (right).*

The size of the buffers (10m and 20 m) were averages from the range of the scanners according to the heat maps. In the end all the triangles belonging to each subspace were merged together and a new shapefile with the resulting eight polygons was created (Figure 73). This method did also result in many overlapping polygons that belonged to different buffers. Manually, these overlapping polygons were deleted, so that in the end each subspace had about the same size. The automatic method can thus better be referred to as semi-automatic.



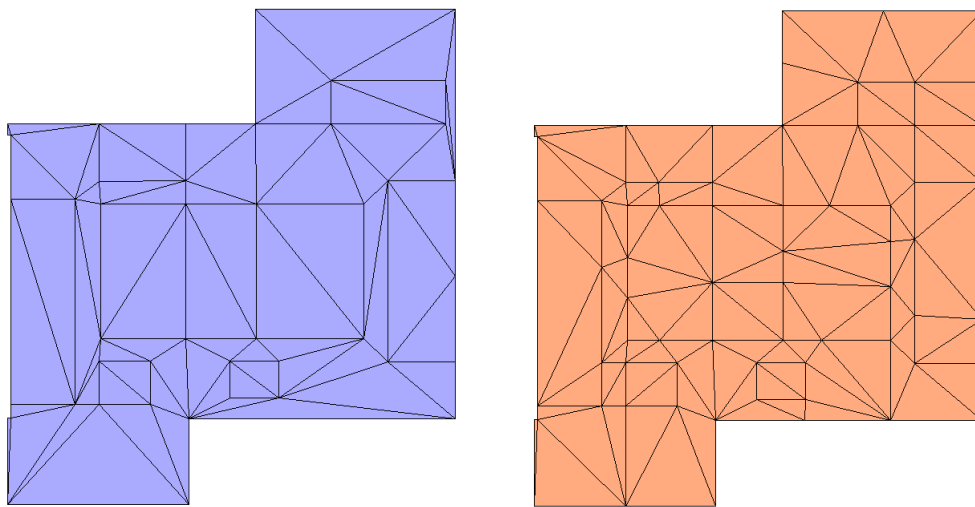
*Fig.73. Automatic space subdivision with eight different subspaces.*





*Fig.74. Testing the automatic method for layout 2*

For layout 2 the buffers of 10 m around the scanners cover almost the whole space. Basically we end up with just four subdivisions (see Figure 74). As stated in the guidelines for the subdivision, this is just too coarse for a subdivision. The semi-automatic method only “works” for the first layout, where the scanners are placed far apart. One solution to solve the problem of the coarse subdivision would be to triangulate the space into much smaller triangles (Figure 75).



*Fig.75. Two different triangulations of the space*

In the figure above it can be seen that the size of the triangles is almost doubled. When using the bigger triangles, the subdivision will become even coarser and has much more overlap, as can be seen in the figure below. Thus an obtained subdivision, can be improved, by using more nodes for the triangulation. By using smaller triangles, the subdivision will become smoother and there will be less overlaps in the subspaces.

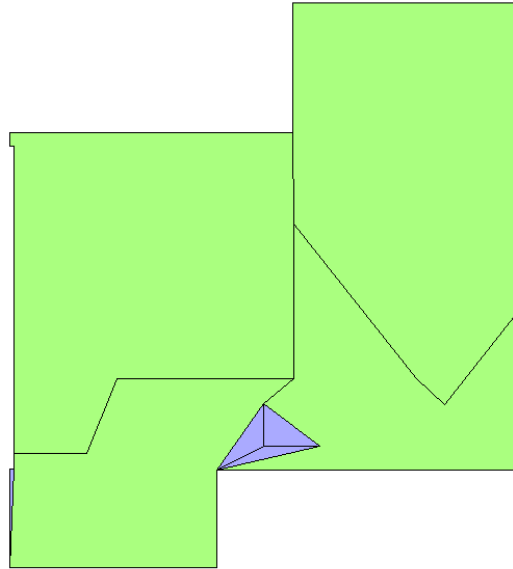


Fig.76. Using bigger triangles, results in an even coarser subdivision

Many different space subdivisions can be developed, but for this project only the intuitive and the semi- automatic space subdivision have been implemented and used in the localization component. The research done by the team, has shown that factors that have an effect on the subdivision are the size of the triangles, the scanner placement or the distance between the scanners and the accuracy of the localization. An automatic subdivision should take these factors into account.

A slightly better result in the localization has been registered with the semi- automatic space subdivision because it takes into account the range of the scanners, which are not considered by the intuitive one. However, the intuitive space subdivision seems in the end to be the most suitable for being used in the navigation, because it is based on the usage of space. By dividing the space into different workspace, is a more understandable way for humans to navigate in an office space. Pros and cons of the two different space subdivisions are summarized in the table below.

Intuitive Space Subdivision	Semi- automatic Space Subdivision
<p><b>PROS:</b></p> <ul style="list-style-type: none"> <li>- Human understandable</li> <li>- Considers the characteristics of the building</li> <li>- Easy to be manually implemented</li> <li>- More suitable for the navigation and visualization</li> </ul>	<p><b>PROS:</b></p> <ul style="list-style-type: none"> <li>- Considers scanner layout and range</li> <li>- Easy to be implemented to multiple floor plans</li> <li>- More precise localization output</li> </ul>
<p><b>CONS:</b></p> <ul style="list-style-type: none"> <li>- Time consuming</li> <li>- Environment must be modelled</li> <li>- Hard to be automatic</li> <li>- More possibility to have invalid polygons</li> </ul>	<p><b>CONS:</b></p> <ul style="list-style-type: none"> <li>- Requires scripting and coding</li> <li>- Not human intuitive subspaces</li> </ul>

Table 5. Pros and cons of the intuitive and automatic space subdivision.

## 10. Localization

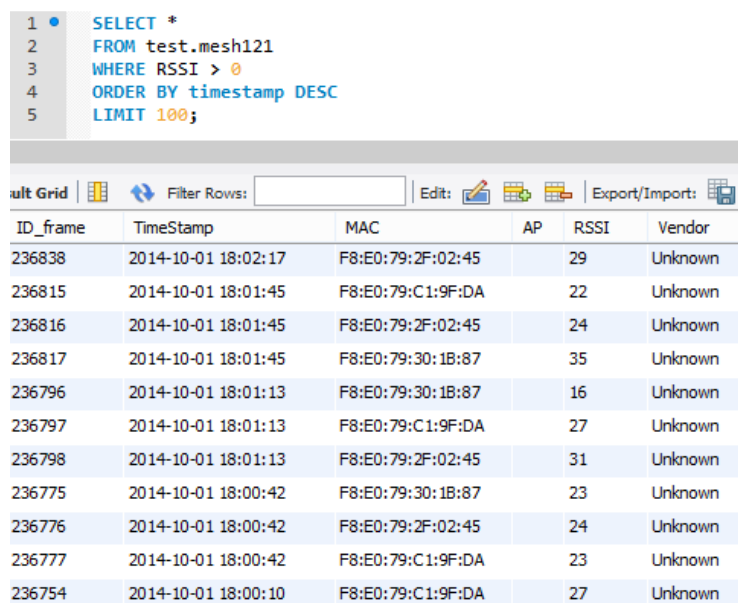
### 10.1 CHOSEN LOCALIZATION METHODS

For all methodologies presented in chapter 6.3, a number of assumptions need to be made in general. If the assumptions are correct, then the feasibility of each method becomes higher and they can be compared for performance.

In our case study, the purpose is to explore how many times a person is localized correctly within a room area defined by the provided subdivision. The area rings approach is over generalizing data and would yield questionable results. The triangulation method would be more applicable if the Angle of Arrival (AoA) was known or if the distances from each scanner would be *precisely* defined using a function that translates RSSI values into distance. Considering the above, the most suitable methods for further exploration are the *multilateration* method and *Wi-Fi fingerprinting*. Below these two methods are described in detail, together with the procedure followed and their formulation of a solution.

### 10.2 MULTILATERATION

The only source of information received by the provided scanners is the RSSI values perceived by each of them. Depending on the scanning interval, each scanner logs each device it detects together with the respective RSSI value in a timely fashion (see Figure 77).



```
1 SELECT *
2 FROM test.mesh121
3 WHERE RSSI > 0
4 ORDER BY timestamp DESC
5 LIMIT 100;
```

ID_frame	TimeStamp	MAC	AP	RSSI	Vendor
236838	2014-10-01 18:02:17	F8:E0:79:2F:02:45		29	Unknown
236815	2014-10-01 18:01:45	F8:E0:79:C1:9F:DA		22	Unknown
236816	2014-10-01 18:01:45	F8:E0:79:2F:02:45		24	Unknown
236817	2014-10-01 18:01:45	F8:E0:79:30:1B:87		35	Unknown
236796	2014-10-01 18:01:13	F8:E0:79:30:1B:87		16	Unknown
236797	2014-10-01 18:01:13	F8:E0:79:C1:9F:DA		27	Unknown
236798	2014-10-01 18:01:13	F8:E0:79:2F:02:45		31	Unknown
236775	2014-10-01 18:00:42	F8:E0:79:30:1B:87		23	Unknown
236776	2014-10-01 18:00:42	F8:E0:79:2F:02:45		24	Unknown
236777	2014-10-01 18:00:42	F8:E0:79:C1:9F:DA		23	Unknown
236754	2014-10-01 18:00:10	F8:E0:79:C1:9F:DA		27	Unknown

Fig. 77. Example of logs by scanner named “mesh121”.

→”Trustability” of values based on time:

In theory, the RSSI values perceived by a scanner of a scanned device in a specific position should not vary much over time. Considering this, one could detect definite changes in these values whenever a device changes position (see Figure 78).

In more detail, when trying to detect a user’s position, a number of RSSI values need to be taken into account. If the request is performed at time = NOW ( $t_0=0$ ), then the algorithm

should “look back” in time at a considerable timebox and see which values are suitable for the averaging. As such, in theory, these are the values that don’t deviate much. If a huge deviation is found, then the algorithm should cut off any values “older” than the last suitable one. By fitting a line curve and specifying a predefined threshold, one could find these RSSI values so as to use for averaging.

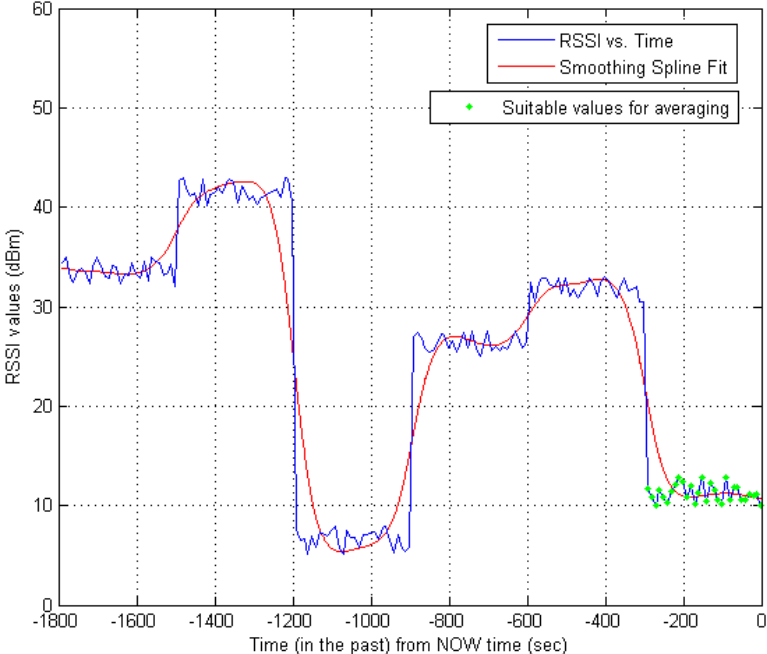


Fig.78. Theoretical distinction between real world positions using RSSI values.

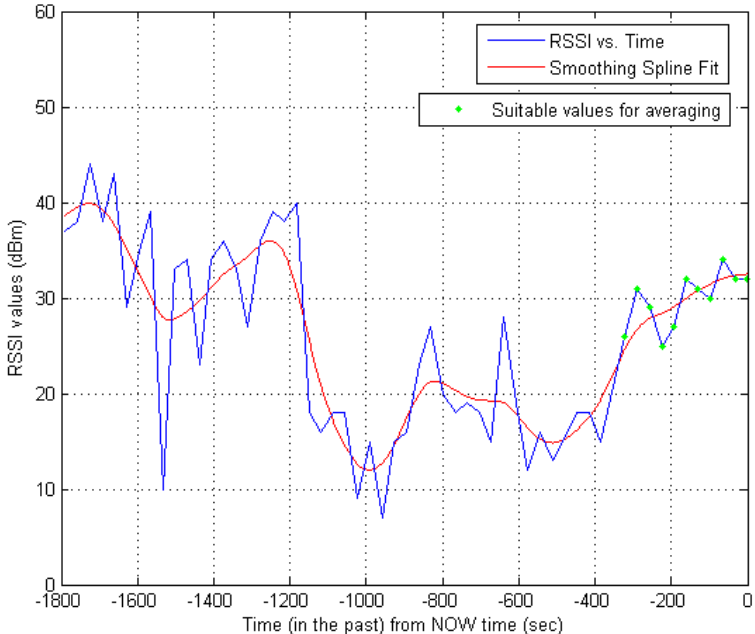


Fig.79. Real RSSI values over time. Normally, 6 ‘distinct’ areas/plateaus should be visible, one every 5 minutes (300sec).

In reality, this type of distinction is difficult, since RSSI values deviate a lot even if the closeness of the real location of the device and the scanner is fixed (Figure 79).

In order to quantify a measure of “trust” for these past values, a logarithmic function was derived that applies a weight to each value, based on how recent it is to the time of request. In particular it is described as such:

$$\text{Trustability} = 1 - (\log_b tdsec)^p$$

where

tdsec: time difference in seconds from  $t_0$  which is ‘now’ (=time of request) and database time logged

$b = \text{interval}_{\text{sec}} + 1$ : time interval in seconds, from the oldest suitable value for averaging until now, plus one

$p = \frac{15}{(\ln \text{interval}_{\text{min}} + 1)^2}$ , with  $\text{interval}_{\text{min}}$  the time interval in minutes, from the oldest suitable value for averaging until now

The above function creates weights (values between 0 and 1) that increase logarithmically the ‘closer’ the value is to current time of request (Figure 80). It is dependent on the timespan that the values cover, thus both the above analysis about choosing suitable values for averaging may apply, but also a set time frame can be used.

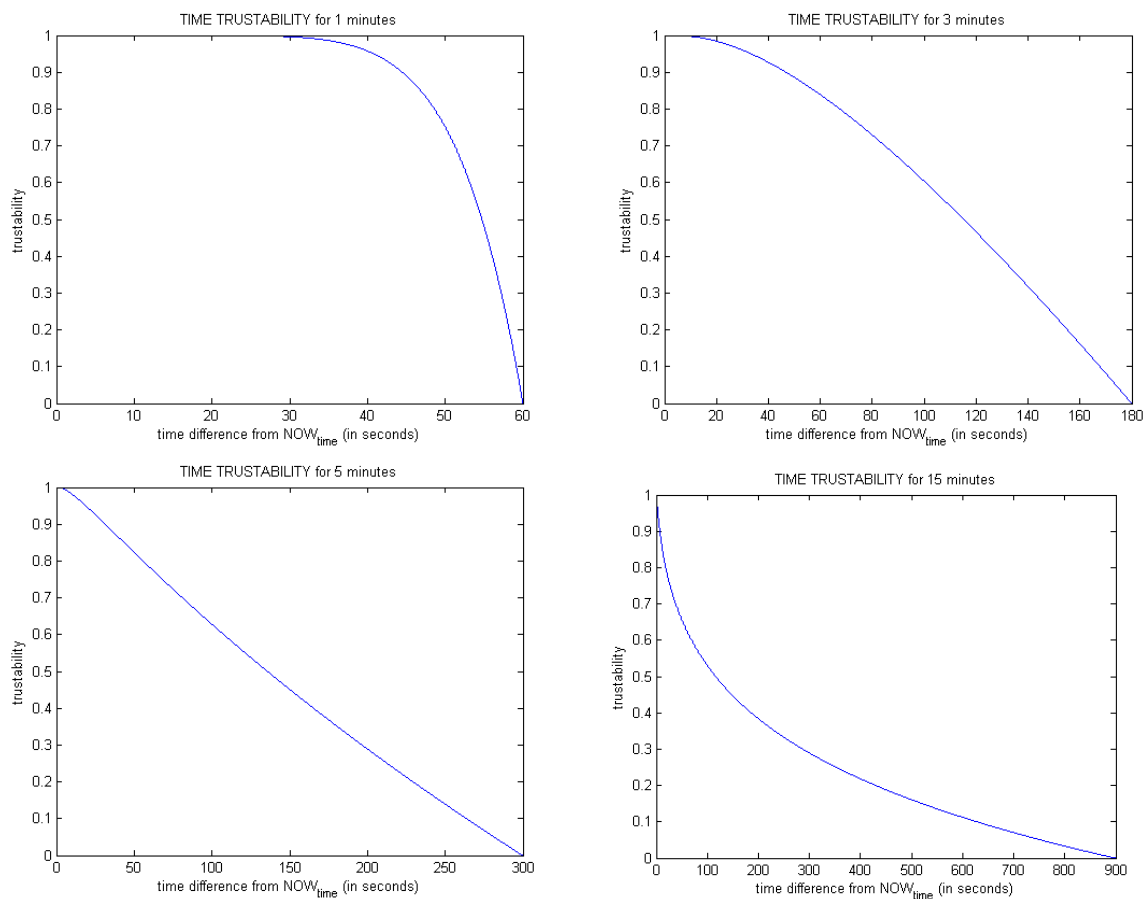


Fig.80. The weights applied to RSSI values change depending on the chosen timespan –the older the values are the less credible they are.

After these weights ( $w_i$ ) are calculated for each RSSI value available ( $RSSI_i$ ), the weighted average of them is computed that yields one RSSI value for each scanner:

$$RSSI_{wavg} = \frac{\sum_{i=1}^m RSSI_i \cdot w_i}{\sum_{i=1}^m w_i}$$

This is the value for which the algorithm proceeds to translate into distance.

### 10.2.1 RSSI into distance:

RSSI translation to distance is not easy to handle and in many cases it has been proven that exact positioning through this method is impossible (A.T. Parameswaran et al, 2009). Nevertheless, in this project a function was constructed to make the multilateration method feasible to an area/room extent. The basis of this function is the log-distance path loss model (Wikipedia: “Log-distance path loss model”, 2014), though a few changes were made. In particular, considering that the RSSI value is 60dBm at 1 meter distance from the scanner, the function is as follows:

$$d = 10^{\frac{R_0 - R}{10n}}$$

Where

$R_0 = 60$  : The RSSI value at 1 meter distance from scanner

$R$  : The retrieved RSSI value

$n = 2.8 + \log_{R_0/2} \frac{R_0}{R}$  : Path-loss coefficient

Normally, the  $n$  coefficient changes depending on the surrounding physical world environment and its value is retrieved empirically or through trial and error. In the current situation, it is developed to be dependent on the RSSI values and logarithmically varies in between values [2.8, 4.0].

Considering the fact that a distance function cannot be perfectly modeled, an added error  $d_{err}$  is taken into account. This error is added and subtracted from computed  $d$  to define the extent to which the the calculated distance is incorrect.

In the end,  $d$  is the value that describes the radius of the circle that encompasses the ring, whose thickness is defined by the  $d_{err}$  calculated and extended to both sides of the the periphery of the circle (see Figure 81). This ring substitutes an area of possible positions for the user to be detected and better localized through the multilateration method.

The error values are also logarithmic and they are larger the smaller the RSSI value is (Figure 82). Thus, the further away from a scanner a device is localized, the greater the possible error, because of the vagueness of the metric.

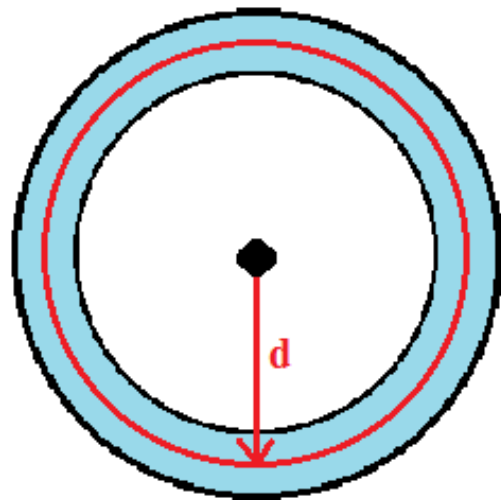


Fig.81. Distance  $d$  from scanner and added error result in a ‘ring’ area.

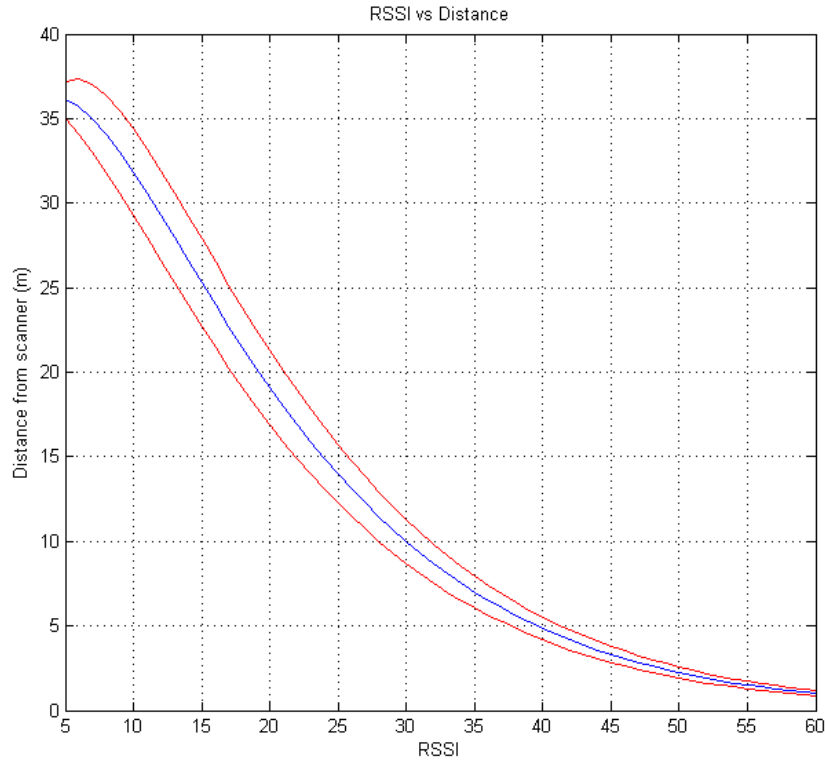


Fig.82. Relation between RSSI and distance, together with the vague error areas.

### 10.2.2 Ring “Importance” Indicator:

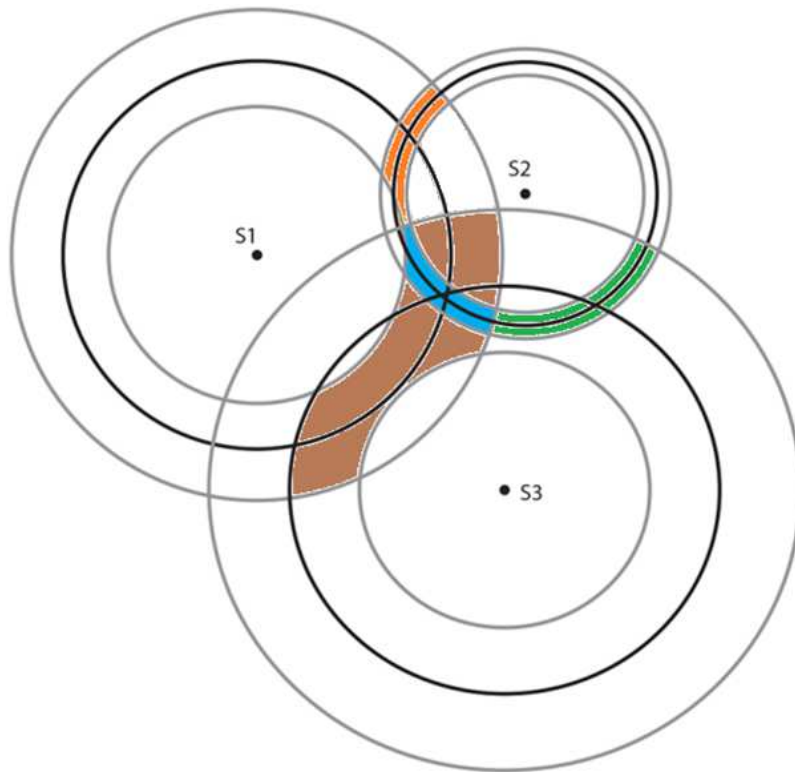
Normally, in trilateration/multilateration methods, at least 3 circles need to intersect in order to achieve localization (see Figure L6). In indoor environments with a small number of scanners this level of availability is quite difficult to achieve. Moreover, in cases where scanner rings intersections describe more than one possible areas, complications arise on which is the most appropriate one to choose. In order to remedy this conundrum an indicator is applied on each ring that defines its priority in choosing it as a best option, even when tri-/multi-lateration is not achieved.

More specifically, this indicator can be a number ( $k_i$ ) that varies depending on the ring’s “importance”, which can be for example its thickness, trustability of  $RSSI_{avg}$  value or another parameterized value. In this project, an indicator is considered that is defined as:

$$k_i = \left( \frac{1}{d_{err} \cdot N} \right)^2$$

where  $d_{err}$  is the error of the circle distance as described earlier and  $N$  the number of scanners. As such, the indicator represents higher values when the error is small, that is when the “thickness” of the ring is small (consequently its distance, thus a high RSSI). The number should vary considerably in order to be distinguishing different rings, therefore the above formula is used.

Each area of a ring is defined by a number  $k_i$ , but when intersections happen between rings, these areas intersections acquire a new  $k_i$  number which is the sum of the indicators involved (example in Figure 83). In this way, the algorithm chooses the area which has the highest indicator to localize the device as the most probable area.



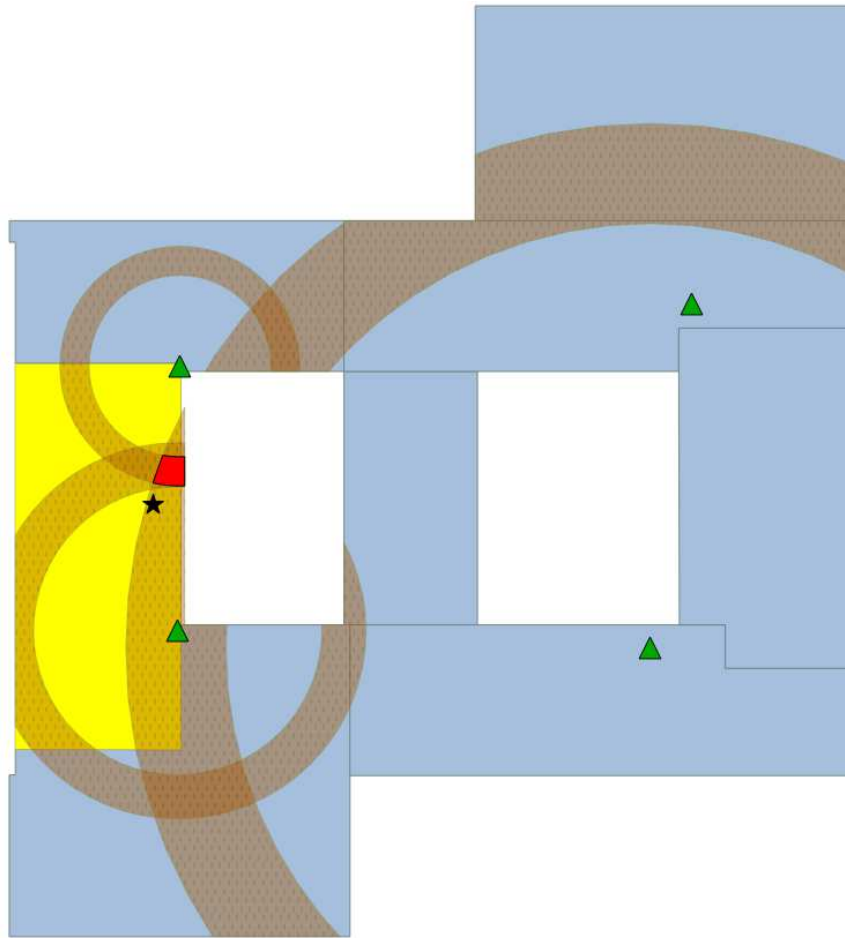
*Fig.83. Calculating the most probable localization area: If the respective indicators for each of the S1,S2,S3 scanners are  $k_1=0.32$ ,  $k_2=0.65$ ,  $k_3=0.14$ , then the colored intersection areas have summed values of  $k_{12}$ (orange)=0.97,  $k_{13}$ (brown)=0.46,  $k_{23}$ (green)=0.79 and  $k_{123}$ (blue)=1.11. Therefore, the combined blue area is the one chosen as the most possible location of the device.*

### 10.2.3. Implementation:

When all of the above is performed, the outcome of the algorithm is an area that localizes the device in question within it. In order to make use of this area, the algorithm runs within the boundaries defined by the location of the scanners, the indoors environment and the space subdivision. Rings exceeding the boundary of the building polygon are logically considered unreachable areas by humans, thus they are excluded from the search of solution. Furthermore, areas like stairs and elevators are also considered “holes” for each floor of the building and they are also not taken into account.

Depending on the space subdivision (see chapter 8), the area may lie within multiple subdivisions of the floor polygon. Then, the subdivision which covers most of this area is the one chosen to be the location of the device and returned as final solution to the user (see Figure 84).

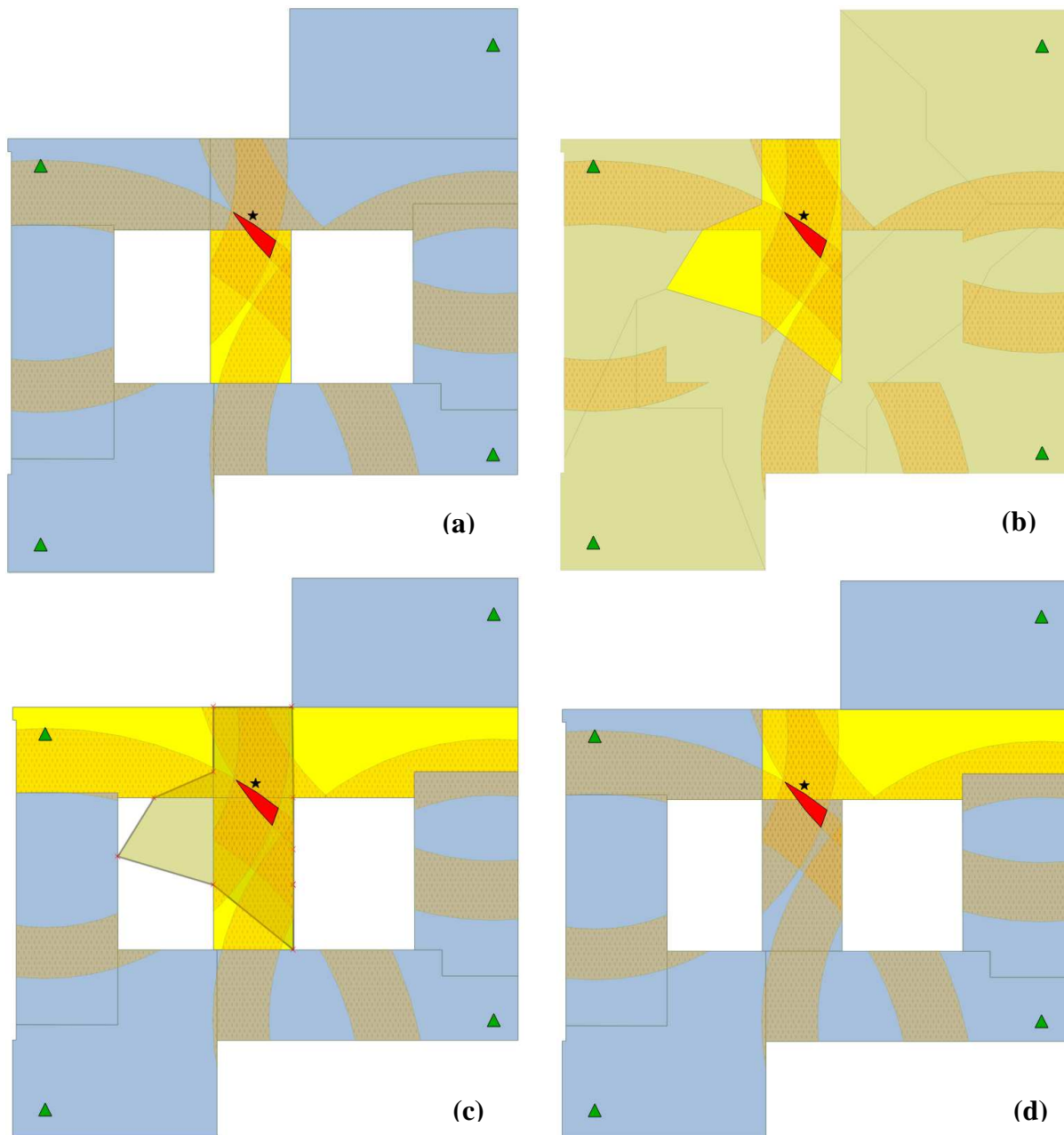




*Fig. 84. Example of localization achieved: three of the scanners (triangle-shaped points) localize the device at different distance. The parts of the rings intersections are evaluated and the most probable location is returned as the area depicted in red. The subdivision polygon (yellow) that intersects the most with this area is returned; in this case 100% of the area lies within one polygon, so this polygon is returned as the device location. The star-shaped point is the ground-truth of the real location of the device, at a distance of about 1.7m from the localized area by the algorithm.*

Another option is to combine different subdivisions in order to achieve localization in at most two adjacent rooms. In more detail, if the intuitive subdivision is the chosen depiction of localization, the algorithm can run delivering a first solution as described above. Afterwards, the algorithm also runs similarly for the automatic subdivision, which takes into account the scanner's positions and range. The subdivision that covers the best the localized area is chosen. This subdivision from the automatic method is imposed on the subdivisions of the intuitive method. The intersections yield a number of areas which relate to a number of subdivisions from the intuitive method. These subdivisions are ordered by maximum area coverage of the automatic subdivision and kept in a list.

After the algorithm returns the first solution (the result taking into account only the intuitive subdivisions) it can also return a second option of an adjacent room that possibly the device lies. This option is the first in order of the computed list that does not match the first solution. As an outcome, the user is given a first solution, but also a suggestion to "search" in the second adjacent room for their target-colleague. The above procedure is explained with the use of an example in Figure 85.



*Fig.85. Example of combined localization: The star-shaped point is the ground-truth position of the device and the red area defines the ring intersections and most probable location area derived by the multilateration algorithm. (a) The largest part of the localized area lies within the central yellow polygon, therefore the algorithm chooses this to be its first solution. (b) The algorithm runs again, but this time for the automatic subdivision. Again the subdivision that intersects the most with the localized area is chosen. (c) This automatic subdivision is intersected with the intuitive subdivisions. The ones that do intersect are kept in a list ordered by area covered (thus most probable locations). (d) Normally, the central polygon would be chosen as a second solution, but since it has already been presented as first in (a), the next polygon on the list is the one lying on the right, thus succeeding in localization. By using this method, the user is given an indication of where to search and then the next most probable area, should the first solution fail.*

#### 10.2.4. Remarks:

The method that describes the underlying algorithm relies heavily on the chosen function which models the translation of RSSI values into distance. The function itself is dependent on the scanner placement and mostly on the surrounding physical environment. Furthermore, the solution can have varying success rate depending on the space subdivision and/or combined solution. Therefore, indoors localization through this process is a multi-layered problem which takes into account a number of factors that are difficult to model, but can achieve a good outcome with a high level of automation and environment modeling.

In the next part the results of using this method for the trial at the ‘De Rotterdam’ building are being presented and analyzed.

#### 10.2.5. Results & Analysis:

The algorithm’s efficiency was tested on the collected data. To describe a performance metric, the first consideration is to check how many of the times the algorithm achieves correct localization within the room the device actually was. In particular, if the ground-truth point of the device lies in a specific subdivision and the algorithm localizes the device in that specific subdivision, this is considered a success.

Furthermore, another case is investigated where localization is achieved on the correct *or* the neighboring subdivisions to the one localized of a building floor. Each subdivision has adjacent ones, therefore if localization is achieved in any one of them, it can still be viewed as a success, considering that two adjacent subdivisions are very close to each other and within a small walkable distance.

Finally, the option of combining subdivisions is explored to make the best use of intuitive and automatic approaches. The first solution is derived using the intuitive subdivision and a second option is provided of the next most possible adjacent room using the automatic subdivision.

In each point the mobile phone devices where placed stable for 5 minutes, therefore the time interval in which the RSSI values are averaged is a priori known and the trustability factor is almost linear (Figure 80). This does not detract from the final results, since the timeframe can be chosen. Furthermore, RSSI values smaller than 5dBm were not taken into account, since they are too small to be considered and their representative distance translation exceeds the scanner’s trustable range.

Layouts 1 and 2 were considered to be the best placements for the scanners; therefore two tests were performed for them.

Each mobile phone device was thus tested on its position and compared to the algorithm position solution. A percentage is given based on how many of the times it is correctly localized in the correct room and/or including neighbours. The tables below summarize this notion:

Layouts Devices	Layout 1: Max Distance		Layout 2: Concrete Square		Layout 3: Elevator pathways	& Layout 4: Half-building
	Test1	Test2	Test1	Test2	Test	Test
f8:e0:79:2f:02:45	63.64	72.73	68.18	63.64	59.09	50.00
f8:e0:79:c1:9f:da	63.64	68.18	54.55	50.00	54.55	50.00
f8:e0:79:30:1b:87	54.55	50.00	63.64	54.55	63.64	58.33
<b>OVERALL</b>	<b>62.12</b>		<b>59.09</b>		<b>59.09</b>	<b>52.78</b>

Table 6. Localization success rate (%) on same room as ground-truth point. If the point lies in the same room as the room returned by the algorithm, it is a success hit.

Layouts Devices	Layout 1: Max Distance		Layout 2: Concrete Square		Layout 3: Elevator pathways	Layout 4: Half-building
	Test1	Test2	Test1	Test2		
f8:e0:79:2f:02:45	95.45	90.91	95.45	95.45	90.91	83.33
f8:e0:79:c1:9f:da	95.45	100.0	86.36	95.45	90.91	91.67
f8:e0:79:30:1b:87	90.91	95.45	90.91	90.91	90.91	83.33
<b>OVERALL</b>	<b>94.70</b>		<b>92.42</b>		<b>90.91</b>	<b>86.11</b>

Table 7. Localization success rate (%) on same room OR neighboring rooms. If the point lies in the same room as the room returned by the algorithm OR any of its adjacent rooms, it is a success hit.

Layouts Devices	Layout 1: Max Distance	
	Test1	Test2
f8:e0:79:2f:02:45	63.64	72.73
f8:e0:79:c1:9f:da	63.64	59.09
f8:e0:79:30:1b:87	59.09	63.64
<b>OVERALL</b>	<b>63.64</b>	

Table 8. Localization success rate (%) on same room as ground-truth point. (automatic subdivision, only layout 1 tested)

Layouts Devices	Layout 1: Max Distance	
	Test1	Test2
f8:e0:79:2f:02:45	90.91	90.91
f8:e0:79:c1:9f:da	95.45	86.36
f8:e0:79:30:1b:87	86.36	86.36
<b>OVERALL</b>	<b>89.39</b>	

Table 9. Localization success rate (%) through combined subdivisions. (intuitive and automatic, only layout 1 tested)

From Table 6 it can be seen that the localization success lies between **59-62%** in the case where the algorithm is required to return exactly 1 answer for localization. When the scanners are placed on half the building the success rate drops to about **53%**. This result implies that having a denser placement of scanners does not necessarily yield better results. Therefore, some research should be performed beforehand in order to take a good decision of where the scanners can be placed. This choice should be made taking into consideration that a high level of discrete separation between readings of the scanners must be achieved.

In Table 7, the neighboring subdivisions are perceived as successes. Of course the result is much higher (**~91-95%**) which shows that the algorithm achieves localization in a close area and not randomly selecting a subdivision. Albeit at the same time, this high rate is a result of the small number of subdivisions in general. If the building floor was subdivided in finer subdivisions, this percentage would hold a higher importance. Again, a lower rate (86%) is portrayed for the half-building scanner layout.

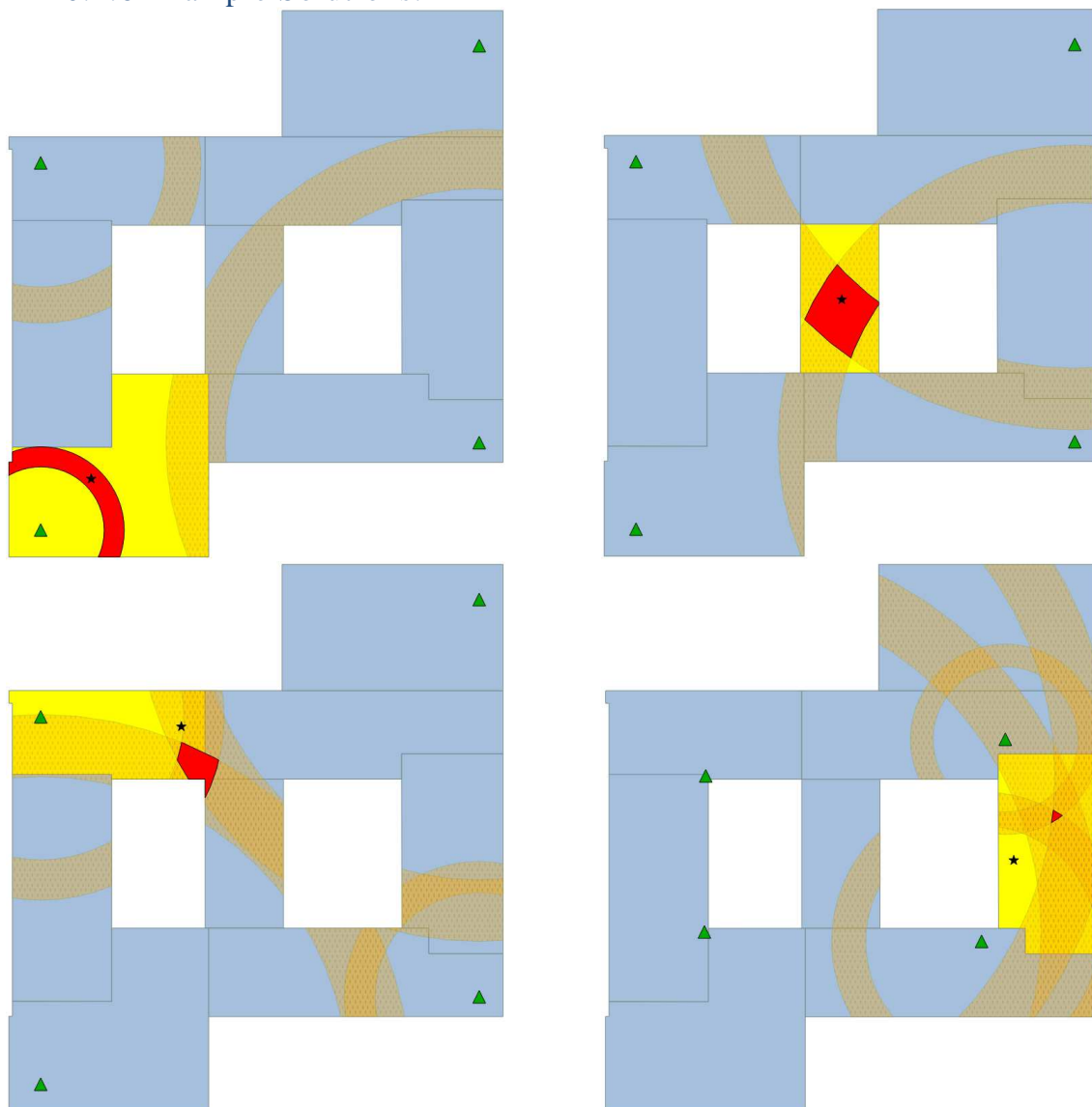
In Table 8, the automatic subdivision is chosen for layout 1 (max-distance scanner placement) to be the layer for localization. In comparison to its intuitive counterpart from Table 5, the

resulting success rate is similar (almost **64%**, compared to 62%). This displays an interesting fact about the algorithm. The successfulness in localizing a device is not as highly dependent on the scanner's range coverage, as long as the whole base area is covered by at least 1 scanner and the subdivision is coarse enough.

On the other hand, if the two perceptions in subdivision are combined, a high success rate can be achieved, up to **89%**, as per Table 9. In this last table, the success is drawn on whether the solution is given on the first option or the second best, which derives from the combination of the intuitive and automatic subdivisions. What needs to be made note of is that this kind of solution is highly understandable for humans and easily followed. If the user that tries to localize their colleague cannot find them on the first solution, the second solution which is simply an adjacent room, is highly possible to result in a success.

From the devices responses a definite result cannot be drawn, yet a slight mention can be made that although all of them where of the same hardware and positioned at the same points during readings, different RSSI values were detected, resulting in differences in distances and localization.

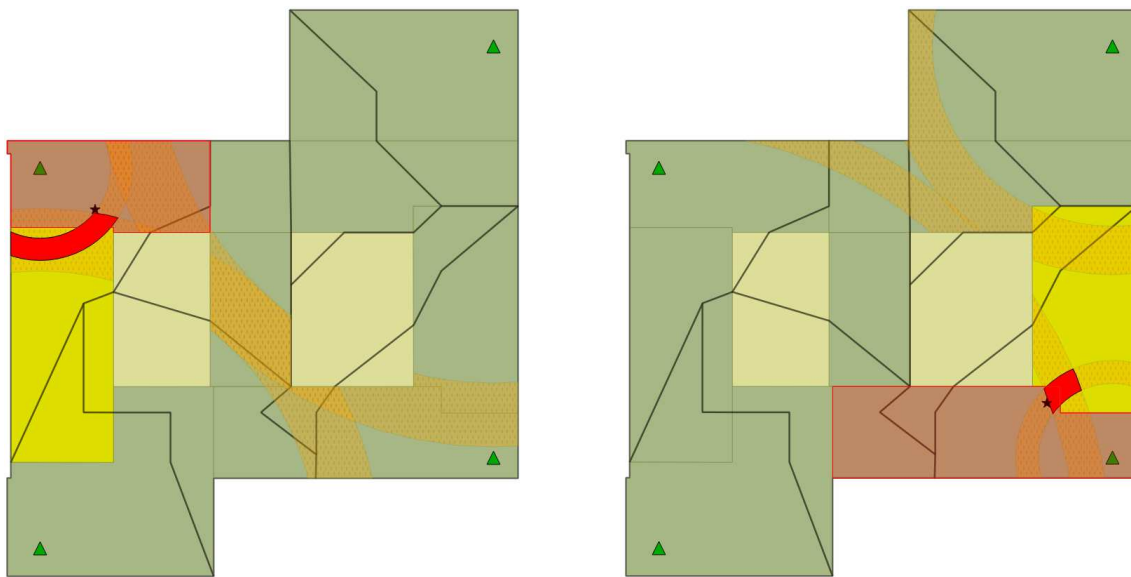
### 10.2.6 Example Solutions:



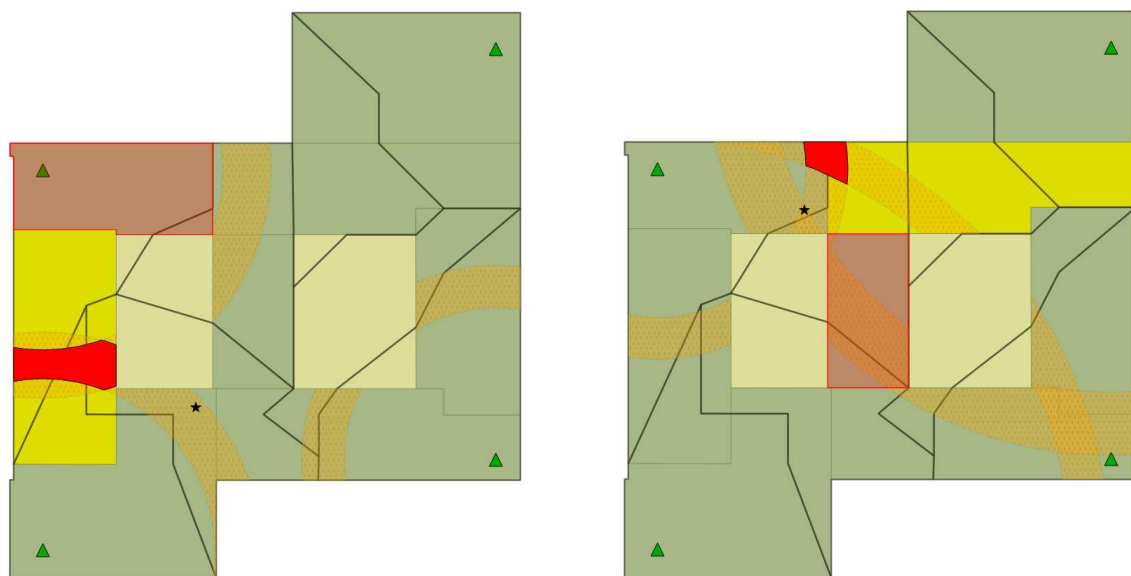
*Fig.86. Successful localization per intersections: 0,2,3 and 4 rings intersecting.*

Above (Figure 86) are presented in visual form some solutions for the trial performed at the ‘De Rotterdam’ building, in order to provide a better understanding behind the running algorithm. Different profiles of localization outcomes are depicted. Normally tri-/multi-lation would require at least 3 rings intersecting, but that would result in a very low success rate and would impede the capabilities of the algorithm, which can localize correctly a device within a room with only data from 1 scanner (1 ring). Nevertheless, because of the “importance” indicator any excessive intersections are discarded and the most suitable one is chosen in every case.

There are though cases where localization is not achieved as a first solution (Figures 87 and 88). Yet, if a combined solution of different subdivisions is used, as described in the implementation part, most of these cases can be treated (Figure 87). The ones that escape treatment (Figure 88) are only about 11%, as shown in the above results.



*Fig.87. Failed localization: treatable cases. First solution (yellow) is wrong, but second suggestion (magenta) succeeds.*



*Fig.88. Failed localization: non-treated cases. Both first and second solutions fail.*

As a final image, some successful localization results are presented for different layouts of scanners in Figure 89:

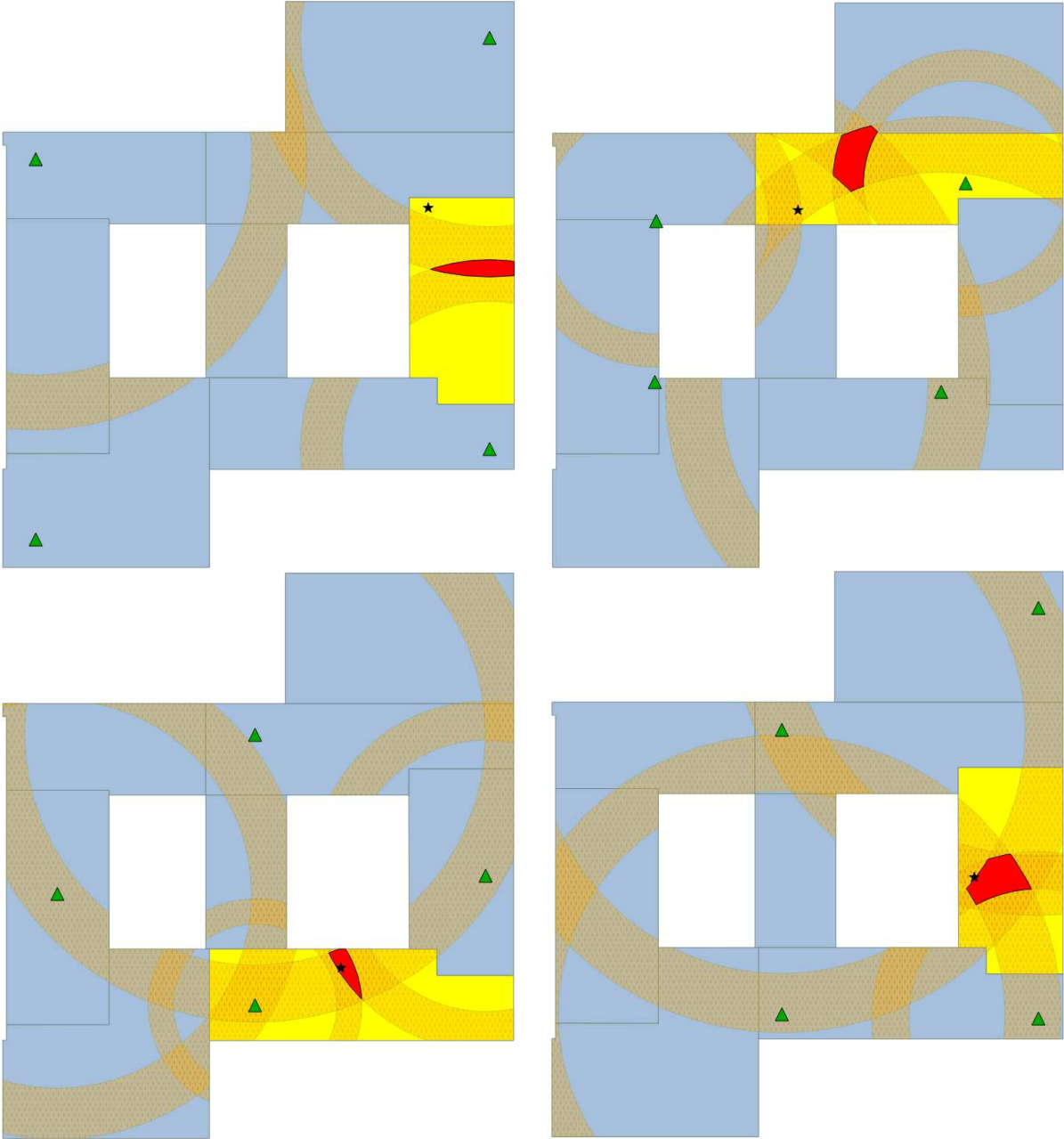


Fig.89. Successful localizations for different scanner layouts.

10.2.7 Conclusions:

The multilateration method as described in the previous sections relies on RSSI values perceived by each scanner for each measured device. RSSI is a vague indicator for distance, therefore a perfect function for translating RSSI values into distance does not exist. Nevertheless, estimations can be made. For this purpose the algorithm constructed can have varying results.

The most important and impeding factor for this translation is the real world physical environment. Walls, materials, noise, multipath, signal propagation and other effects can have a large impact on the results. If this environment can be modeled, a much better result is expected.

Furthermore, the method described is dependent on the space subdivision. If the subdivisions are too small, a lower success rate is expected, but when the localization is achieved, the solution is more precise to the target's real location. If the subdivisions are too large, there will only be a few that cover the whole area of interest. As such, the localization success rate would rise, but the localization area might be too big for a human to perceive it as helpful for navigation.

Finally, results are always dependent on the hardware used. Different devices can yield a different outcome. What could be improved on this behalf is the scanning rate by which the scanners scan their cover area. With more values a better average can be achieved.

For the purposes of the current application, a “middle”-ground solution is given. Localization is achieved within subdivisions that can be described as intuitive and navigable for humans. These subdivisions vary in size (85m<sup>2</sup> to 212m<sup>2</sup>) but span across only a few meters and given the structure of ‘De Rotterdam’ building, they are both suitable for the algorithm and a human to follow. The physical environment could not be modeled, but the results are interesting for a deterministic model. **6 out of 10 times** a device is correctly localized within a subdivision and that result rises to **9 out of 10 times** success, when the next possible subdivision is included in the solution.

These results can be deemed both satisfactory and expected. Indoors localization is achieved usually correctly and a second suggestion allows for verification for success. As such, the application can display to the user the most probable area the target lies and then give a suggestion in case the target is not immediately found by the user. The high level of intuitiveness that such result displays is what makes this method successful.

### 10.3. FINGERPRINTING

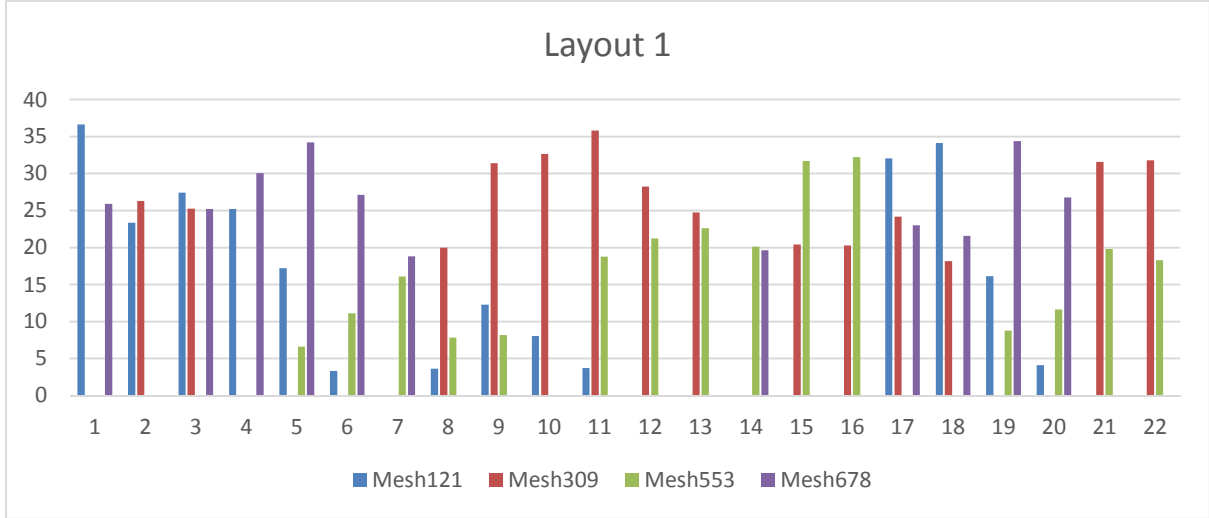
#### 10.3.1 Data preprocessing

As is stated previously, in the tests made in “De Rotterdam” building, four layouts of the four Meshlium scanners were designed for testing focusing on different aspects: max distance, concrete square, elevator and pathways, and half-building. Raw data were collected for each layout with three Samsung smartphones at 22 locations spreading over the floor for five minutes each location. From the fingerprints got from these four tests, the team found layout 3 leaves large vacuums over the whole area where the phone cannot be detected by two or more scanners at the same time, which is not optimistic for Wi-Fi fingerprinting since these vacuum areas cannot be effectively discriminated. On the contrary, layout 4 proved itself incompetent by covering the half building so well that fingerprints at many locations are too similar to be accurately distinguished. As for layout 1 and layout 2, both of them present nice coverage with adequate variations between locations. Thus, the team took same test again for both layout 1 and layout 2 to be more accurate to further research which layout of the scanners is better for Wi-Fi fingerprinting.

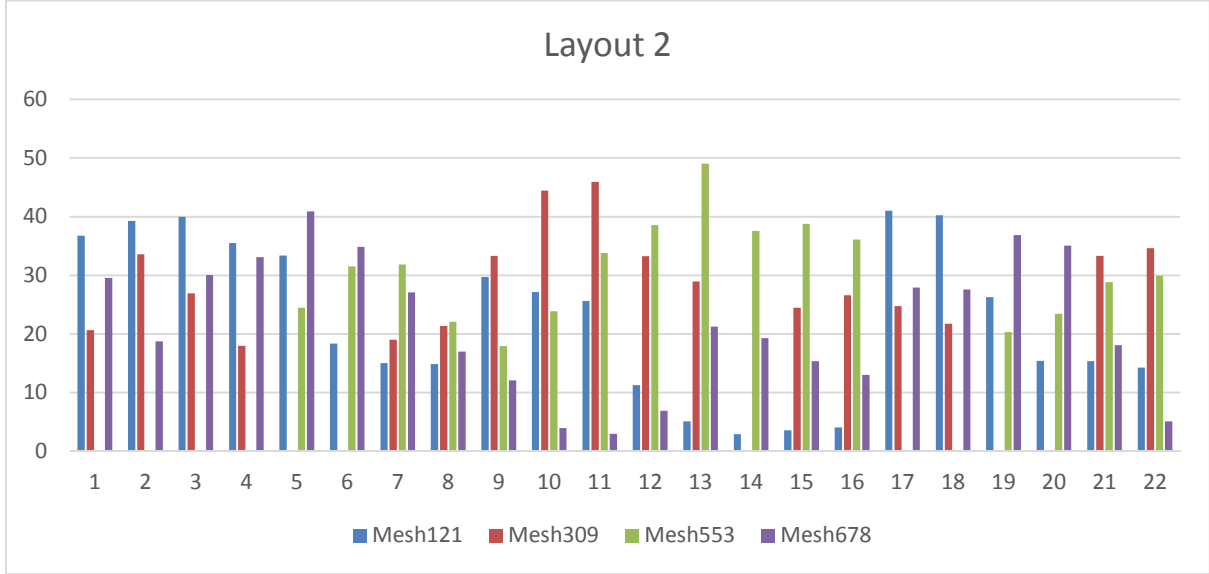
Before the raw data can be actually used in the Wi-Fi Fingerprinting, some preprocessing need to be carried out. First, for each scanner, the RSSI values of these three phones at each locations are all together averaged and the total times they were detected by the scanner are



calculated (see Appendix I). Data collected at locations where the three phones were scanned less than ten times in total in the five minutes (i.e. each phone was scanned three times or less) are considered unreliable and marked yellow. This happens when the phones are too far and almost beyond the reach of the scanner or there are a lot of obstructions between them. From Appendix II, it can also be seen that data marked yellow from the first and the second test always vary a lot, which with each other except for those zero values with zero scan which indicates the phones were not detected by the scanners at all. Then, to derive the final RSSI values to be used, the averaged data from the two tests for each layout are averaged again and values are set to be zero wherever either of the data from the two tests is marked yellow (see Appendix II). Hereby, the fingerprints of the four scanners at each location are obtained, which is shown in the charts below.



(a)



(b)

Fig.90. Fingerprints: (a) layout 1; (b) layout 2

The last step before interpolation is to define a local coordinate system for these sampling points, which is usually realized by overlapping a regular grid onto the area. The team chose a grid of 2 by 2 squares and a grid of 4 by 4 squares to research what would be the best size of the cells. These two grids are shown in Figure 91.

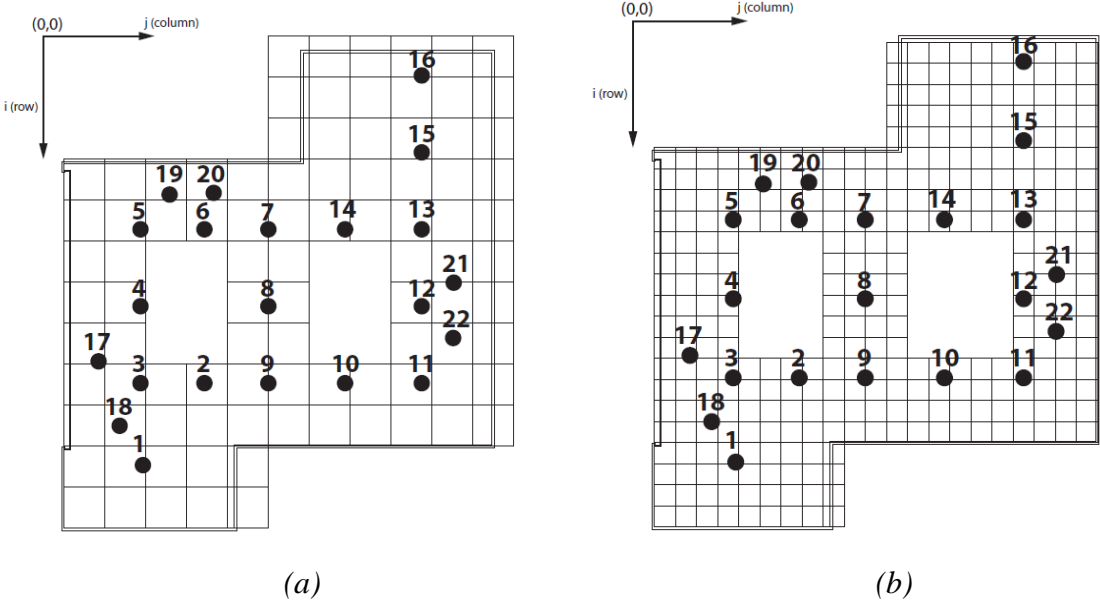


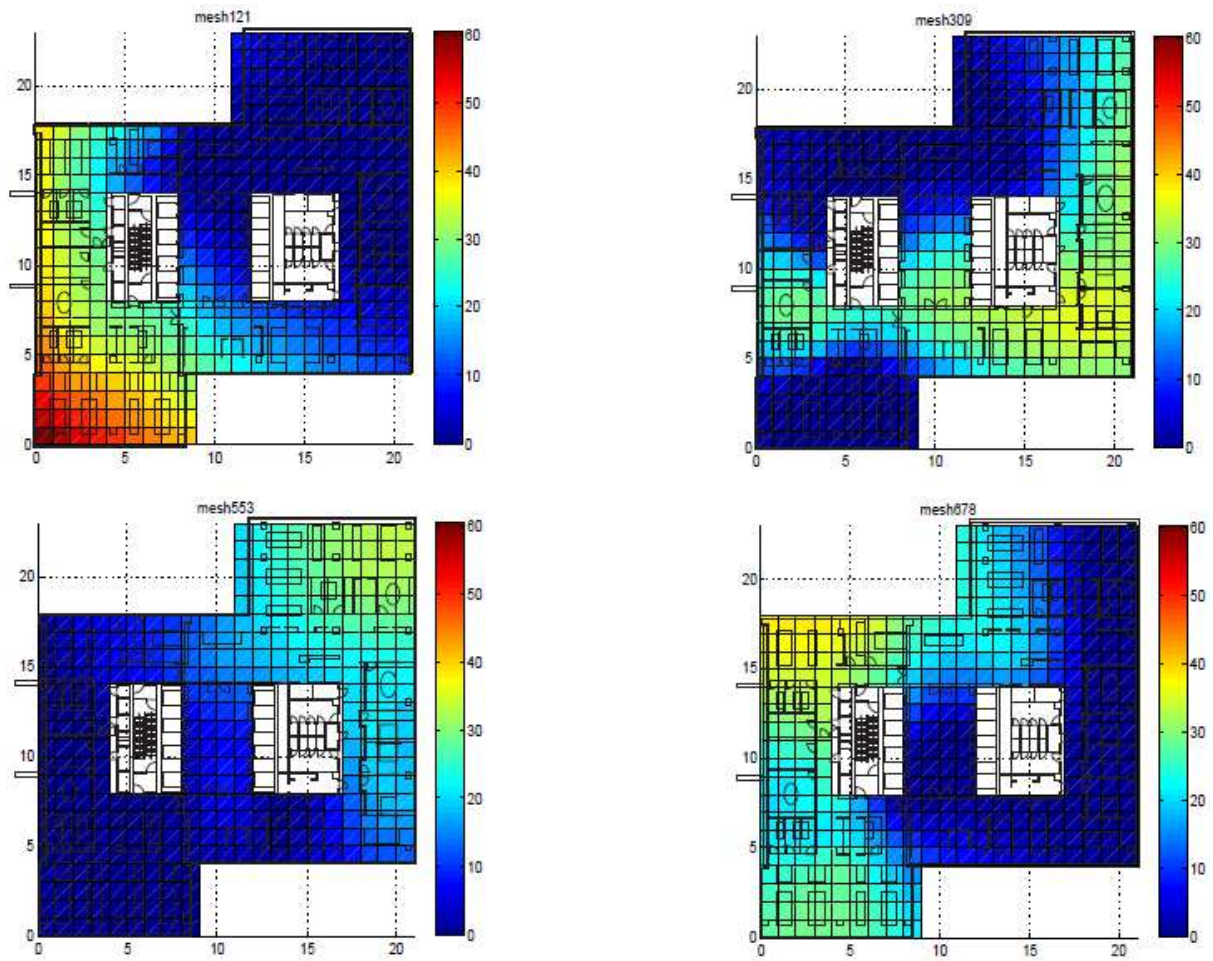
Fig.91. Sampling points in different grids: (a) 4 by 4 ; (b) 2 by 2

Then, the coordinates of these 22 points in the two different grids are obtained, which are shown in Appendix III together with the according average RSSI values as the final input data.

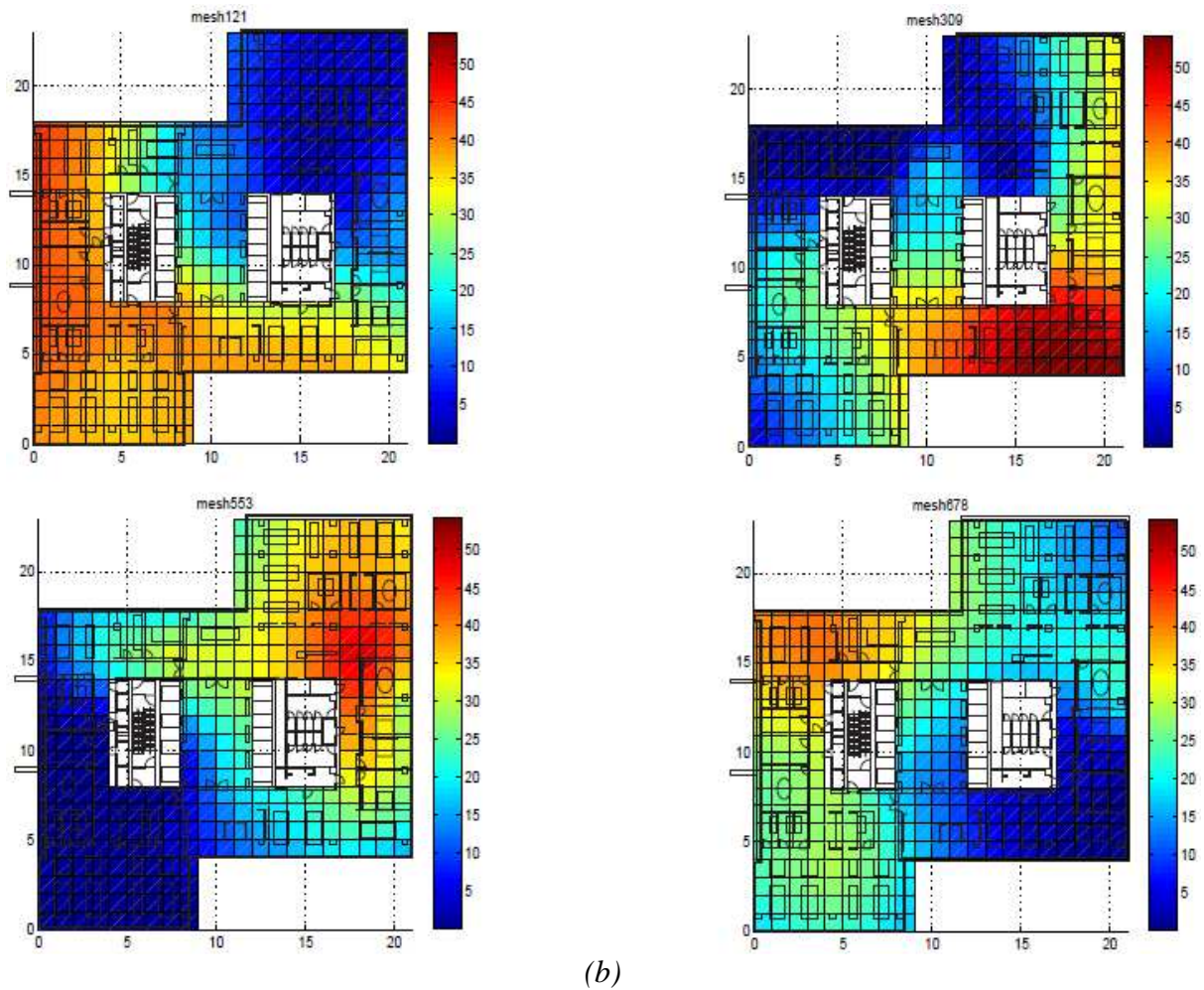
10.3.2. Interpolation

Out of the 22 points, 15 points are taken out to do the interpolation to create the heat maps of each scanner for each layout, while the other 7 points are testing points to check the accuracy of these heat maps.

The interpolation is calculated in Python using ‘scipy.interpolate.Rbf’, a function for interpolation of n-dimensional scattered data of radial basis function, since the signal of a scanner goes out radially in open area in theory. The heat maps are shown in Figure 92 (a&b).



(a)



(b)  
Fig.92. Heatmaps: (a) layout 1; (b) layout 2

### 10.3.3 Matching

As is stated before, there are two matching algorithms. For this project, the Nearest Neighbor method is chosen to match between the recorded fingerprints in the heatmaps and the seven live fingerprints. Location with the least sum of squared differences is assumed to be the best match.

$$R_{min} = MIN \left( \sqrt{\sum_{i=1}^n (R_i - FP_i)^2} \right)$$

Table 10 shows the result of each layout with each grid.  $(x_0, y_0)$  are the coordinates of the square which the ground truth lies in, while  $(x, y)$  is the calculated coordinates of the best match point.  $dx$  and  $dy$  are the difference between these two sets of coordinated.  $R_{min}$  is the minimal sum of squared differences between the recorded and live fingerprints that comes along with the best match.  $Dist$  is the overall distance error between the ground truth and the best match point, which is calculated by these functions below:

$$Dist = L \times \sqrt{dx^2 + dy^2}$$

$L=2m$  when using 2 by 2 grid,  $L=4m$  when using 4 by 4 grid

Table 10. Heat maps testing results

No.	$x_0$	$y_0$	x	y	dx	dy	Dist (m)	$R_{min}$ (dBm)
3	15	3	14	1	1	2	4.47213595499958	5.239318735613585
6	8	6	7	7	1	-1	2.8284271247461903	4.526167534693688
10	15	13	14	8	1	5	10.198039027185569	8.399138514819178
15	4	17	0	17	4	0	8.0	0.550764724593878
18	18	2	16	2	2	0	4.0	2.3427934329267632
20	6	7	7	7	-1	0	2.0	3.618259832854877
21	11	19	12	19	-1	0	2.0	0.6403581380151838
AVG Dist (m)							4.785515	

(a) Layout 1 with 2 by 2 grid

No.	$x_0$	$y_0$	x	y	dx	dy	Dist (m)	$R_{min}$ (dBm)
3	8	1	7	0	1	1	5.656854249	5.239318736
6	4	3	4	3	0	0	0	5.252243749
10	8	6	7	4	1	2	8.94427191	8.618502638
15	2	8	0	8	2	0	8	0.550764725
18	9	1	8	0	1	1	5.656854249	4.131643285
20	3	3	4	3	-1	0	4	4.534343626
21	6	9	6	9	0	0	0	0.669422414
AVG Dist (m)							4.60828292	

(b) layout 1 with 4 by 4 grid

No.	$x_0$	$y_0$	x	y	dx	dy	Dist (m)	$R_{min}$ (dBm)
3	15	3	14	2	1	1	2.8284271247461903	2.550717802348352
6	8	6	5	8	3	-2	7.211102550927978	8.7278583641188
10	15	13	16	19	-1	-6	12.165525060596439	5.10795213494923
15	4	17	1	17	3	0	6.0	2.537312856679257
18	18	2	16	0	2	2	5.656854249492381	1.593611087663197
20	6	7	5	7	1	0	2.0	7.060631555670865
21	11	19	14	11	-3	8	17.08800749063506	10.483064541304355
AVG Dist (m)							7.564274	

(c) layout 2 with 2 by 2 grid

No.	$x_0$	$y_0$	x	y	dx	dy	Dist (m)	$R_{min}$ (dBm)
3	8	1	7	1	1	0	4	2.2026924
6	4	3	3	3	1	0	4	11.113585
10	8	6	9	9	-1	-3	12.649111	6.0042854
15	2	8	1	8	1	0	4	4.1278081
18	9	1	9	0	0	1	4	1.9830051
20	3	3	3	3	0	0	0	10.211058
21	6	9	7	5	-1	4	16.492423	10.57897
AVG Dist (m)							6.44879	

(d) layout 2 with 4 by 4 grid

From the results, it can be seen that layout 1 with 4 by 4 grid provides the best accuracy. To be more specific, layout 1 outperforms layout 2 with either 2 by 2 or 4 by 4 grid and 4 by 4 grid is better than 2 by 2 grid in either layout 1 or layout 2. The reason behind this might be that the fingerprints generated in layout 1 are better detected since there is more vacuum area in layout 1 than that in layout 2. Besides, the granularity of 2 by 2 grid might be too small for either the interpolation to accurately simulate the fingerprints in real environment or the matching algorithm to effectively distinguish the differences between the squares.

#### 10.3.4 Result with space subdivision

Table 11 shows the localization result of these 4 different set-ups, namely layout 1 with 2 by 2 grid, layout 1 with 4 by 4 grid, layout 2 with 2 by 2 grid and layout 2 with 4 by 4 grid, when fingerprinting is combined with space subdivision.  $ID_{p0}$  is the index in the grid of the square which the ground truth lies in, while  $ID_p$  is the index of the outcome square.  $ID_{sub0}$  indicates the subdivision which the ground truth lies in, while  $ID_{sub}$  indicates the subdivision the point is actually assigned to.

Table 11. Localization result in space subdivision

No.	$ID_{p0}$	$ID_p$	$ID_{sub0}$	$ID_{sub}$	Final result
3	150	169	2	2	'localized successfully'
6	300	322	3	3	'localized successfully'
10	160	176	4	0	'localization failed'
15	395	479	7	7	'localized successfully'
18	86	128	1	2	'localization failed'
20	343	322	3	3	'localized successfully'
21	250	229	5	5	'localized successfully'

(a) layout 1 with 2 by 2 grid

No.	$ID_{p0}$	$ID_p$	$ID_{sub0}$	$ID_{sub}$	Final result
3	34	44	4	0	'localization failed'
6	80	80	3	3	'localized successfully'
10	39	48	5	5	'localized successfully'
15	107	129	7	7	'localized successfully'
18	23	33	1	1	'localized successfully'
20	91	80	3	3	'localized successfully'
21	64	64	0	0	'localized successfully'

(b) layout 1 with 4 by 4 grid

No.	$ID_{p0}$	$ID_p$	$ID_{sub0}$	$ID_{sub}$	Final result
3	150	170	2	2	'localized successfully'
6	300	365	3	6	'localization failed'
10	160	145	4	4	'localized successfully'
15	395	458	7	7	'localized successfully'
18	86	126	1	2	'localization failed'
20	343	364	3	3	'localized successfully'
21	250	179	5	0	'localization failed'

(c) layout 2 with 2 by 2 grid

No.	$ID_{p0}$	$ID_p$	$ID_{sub0}$	$ID_{sub}$	Final result
3	34	45	4	0	'localization failed'
6	80	91	3	3	'localized successfully'
10	39	31	5	2	'localization failed'
15	107	118	7	7	'localized successfully'
18	23	22	1	1	'localized successfully'
20	91	91	3	3	'localized successfully'
21	64	49	0	5	'localization failed'

(d) layout 2 with 4 by 4 grid

From the table it can be noticed that layout 1 with 4 by 4 grid has the best successful rate (6/7) of localization in subdivision among these 4 set-ups, while those for layout 2 with either 2 by 2 or 4 by 4 grid are both 4/7. Between them is layout 1 with 2 by 2 grid with a successful rate of 5/7. This also complies with the outcome in Table 9.

Examples of the localization result are shown in Figure 93, in which the red square is the location of the ground truth while the blue square is the localized squared. Figure 93 (a) shows the point is successfully localized in the same subdivision which the ground truth lies in; Figure 93 (b) shows another case where the point is wrongly localized in an adjacent subdivision of the one in which it actually should be.

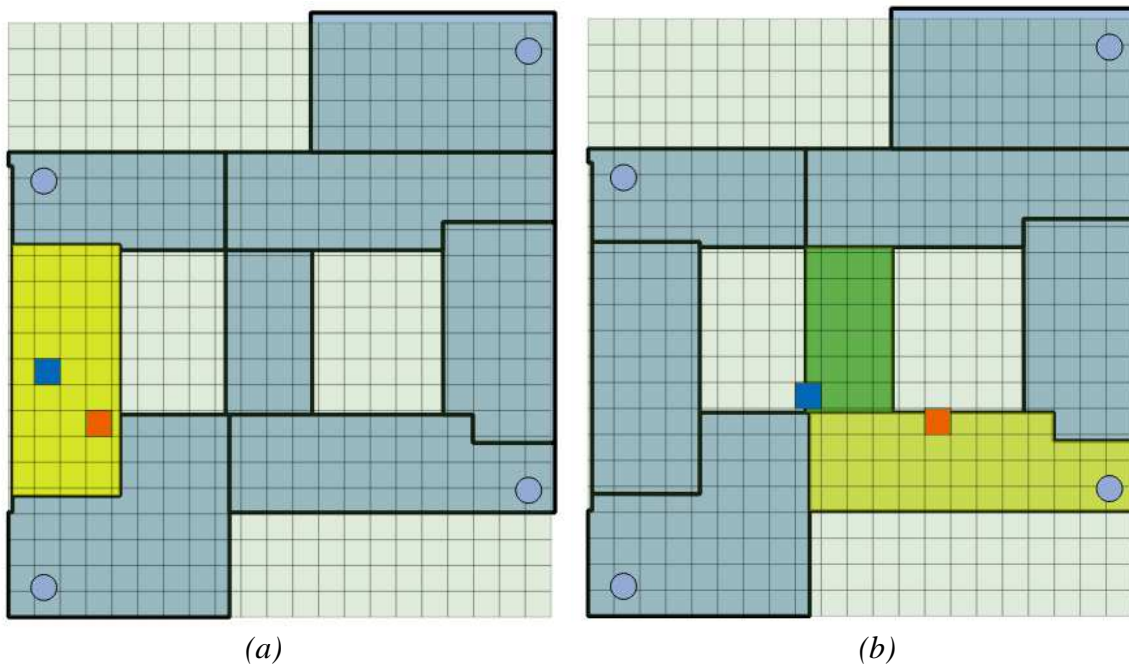


Fig. 93. Examples of localization result of layout 1 with 2 by 2 grid in subdivision (a) Successful localization; (b) Failed localization

### 10.3.5 Improved result

Here, similarly, the final localization result can be also improved with the algorithm introduced above, which utilizes another possible space subdivision that is based on the coverage area of the signal of each scanner to give other possible options to the user. Table 12 shows how the localization can be improved when given a second or even a third option for the 4 different set-ups.

Table 12. Improved localization result in space subdivision

No.	ID <sub>p0</sub>	ID <sub>p</sub>	ID <sub>sub0</sub>	ID <sub>sub1</sub>	ID <sub>sub2</sub>	ID <sub>sub3</sub>	Final result
3	34	44	4	0	1	4	'localized successfully'
6	80	80	3	3	None	None	'localized successfully'
10	39	48	5	5	None	None	'localized successfully'
15	107	129	7	7	None	None	'localized successfully'
18	23	33	1	1	None	None	'localized successfully'
20	91	80	3	3	None	None	'localized successfully'
21	64	64	0	0	None	None	'localized successfully'

(a) layout 1 with 4 by 4 grid

No.	ID <sub>p0</sub>	ID <sub>p</sub>	ID <sub>sub0</sub>	ID <sub>sub1</sub>	ID <sub>sub2</sub>	ID <sub>sub3</sub>	Final result
3	150	169	2	2	None	None	'localized successfully'
6	300	322	3	3	None	None	'localized successfully'
10	160	176	4	0	1	4	'localized successfully'
15	395	479	7	7	None	None	'localized successfully'
18	86	128	1	2	1	None	'localized successfully'
20	343	322	3	3	None	None	'localized successfully'
21	250	229	5	5	None	None	'localized successfully'

(b) layout 1 with 2 by 2 grid

No.	ID <sub>p0</sub>	ID <sub>p</sub>	ID <sub>sub0</sub>	ID <sub>sub1</sub>	ID <sub>sub2</sub>	ID <sub>sub3</sub>	Final result
3	34	45	4	0	6	3	'3 chances at most, localization failed'
6	80	91	3	3	None	None	'localized successfully'
10	39	31	5	2	1	None	'no other choice, localization failed'
15	107	118	7	7	None	None	'localized successfully'
18	23	22	1	1	None	None	'localized successfully'
20	91	91	3	3	None	None	'localized successfully'
21	64	49	0	5	4	None	'no other choice, localization failed'

(c) layout 2 with 4 by 4 grid

No.	ID <sub>p0</sub>	ID <sub>p</sub>	ID <sub>sub0</sub>	ID <sub>sub1</sub>	ID <sub>sub2</sub>	ID <sub>sub3</sub>	Final result
3	150	170	2	2	None	None	'localized successfully'
6	300	365	3	6	0	3	'localized successfully'
10	160	145	4	4	None	None	'localized successfully'
15	395	458	7	7	None	None	'localized successfully'
18	86	126	1	2	3	6	'3 chances at most, localization failed'
20	343	364	3	3	None	None	'localized successfully'
21	250	179	5	0	6	3	'3 chances at most, localization failed'

(d) layout 2 with 2 by 2 grid

From the following table (Table 13), it can be seen that when given a second option, one more point is localized successful in layout 1 with 2 by 2 grid, while rates for the other set-ups stay the same. When given a third option, the successful rate of both set-ups of layout 1 with either



2 by 2 or 4 by 4 grid increases to 7/7, while that of layout 2 with each kind of grid has also improved in different degree, going up to 5/7 and 6/7 respectively.

Successful rate	Layout 1		Layout 2	
	2 by 2	4 by 4	2 by 2	4 by 4
Options				
1 <sup>st</sup>	5/7	6/7	4/7	4/7
1 <sup>st</sup> +2 <sup>nd</sup>	6/7	6/7	4/7	4/7
1 <sup>st</sup> +2 <sup>nd</sup> +3 <sup>rd</sup>	7/7	7/7	5/7	6/7

*Table 13. Changes of successful rate for different set-ups*

### 10.3.6 Conclusion

The Wi-Fi fingerprinting method uses the unique profile of RSSI values in each square to search for the best match between the recorded fingerprints from the heat maps of each scanner created by interpolating the sampling points and the live fingerprints provides by the application. It is more deterministic and also more accurate than other existing indoor localization methods, while it requires heavy labor and large amount of time for collecting the training data. This method relies heavily on its surrounding environment since every heatmap it uses is created under a specific environment. Every time the environment changes, the training data must be recollected to update the heat maps, which in turn greatens the burden in the training phase.

Furthermore, the accuracy of this method also depends on the granularity of the grid it uses to define the coordinate system and the space subdivision applied after a certain square of the grid that a target is localized in is returned. If the granularity of the grid is too coarse, the localization will be meaningless since the given area will never be accurate enough for either finding people or for navigation. However, the results shown above also prove that too small granularity of the grid can endanger the overall localization successful rate, either with or without a space subdivision applied, maybe because it is not suitable for either the interpolation to accurately simulate the fingerprints in real environment or the matching algorithm to effectively distinguish the differences between the squares.

Two kinds of space subdivision are used in this project: one with higher intuitiveness and one that is closer to a theoretical space subdivision since it is based on the coverage area of the signal of each scanner. An algorithm is applied that uses the intuitive subdivision as a front interface to interact with the user and the theoretical one as a background layer to perfect the localization.

For a more precise localization, enough sampling points must be guaranteed either for interpolation or testing. In the tests taken in 'De Rotterdam' building for this project, the team only collected data at 22 points. 15 of them are used in the interpolation, while the other 7 are used to test the localization. On one hand, 15 points are far from enough for accurate interpolation to create heatmaps that can perfectly simulate the real environment; on the other hand, 7 testing points are also nowhere near persuasive and convincing to draw any solid conclusions from the result. Besides, the time period for collecting data at each location needs to be controlled precisely. If during the timespan when a phone should be collecting data at certain location the phone is actually still moving, there will be some deviations in the raw data that might possibly compromise the accuracy of the final result.

## 11. Navigation

In order to enable the user to find their colleague, navigation is needed to communicate the route to the user. For accomplishing to this task, a description of how the subdivided spaces are connected is needed.

### 11.1 IMPLEMENTATION

#### 11.1.1 Deriving the network

After having analysed the main approaches for generating the navigation system, the network approach was selected for 'De Rotterdam' building. The reasons why this method was preferred are the following:

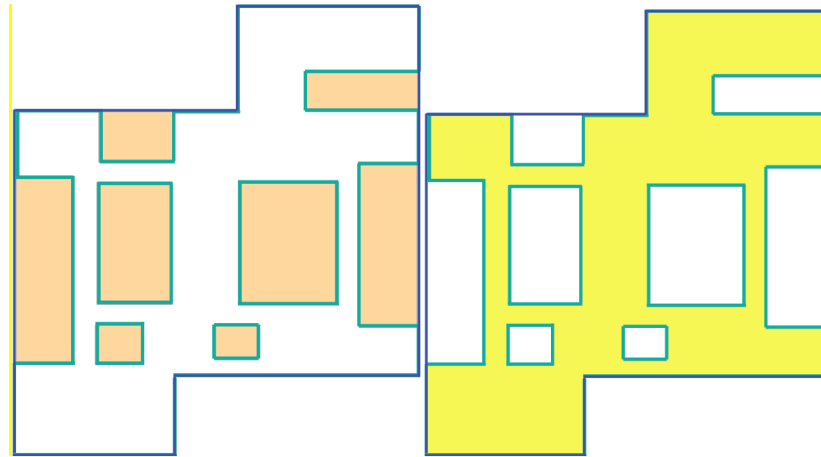
- It fits better with the characteristics of 'De Rotterdam' building: corridor around the concrete block where the facilities are located (elevators, stairs, toilets, etc.) and open spaces around it.
- High positional accuracy is not needed since employees can easily find the colleagues in open spaces and through glass walls.
- Easy to design since just few nodes and edges are needed for navigating through the building.

The simplest way of designing a network is to do so manually for each floor plan or building part. The drawbacks of this method are that it is time consuming and it has to be repeated for changes in the space subdivision. A generalized method of generating a network will not only save time, but may also be practical in other buildings than 'De Rotterdam'.

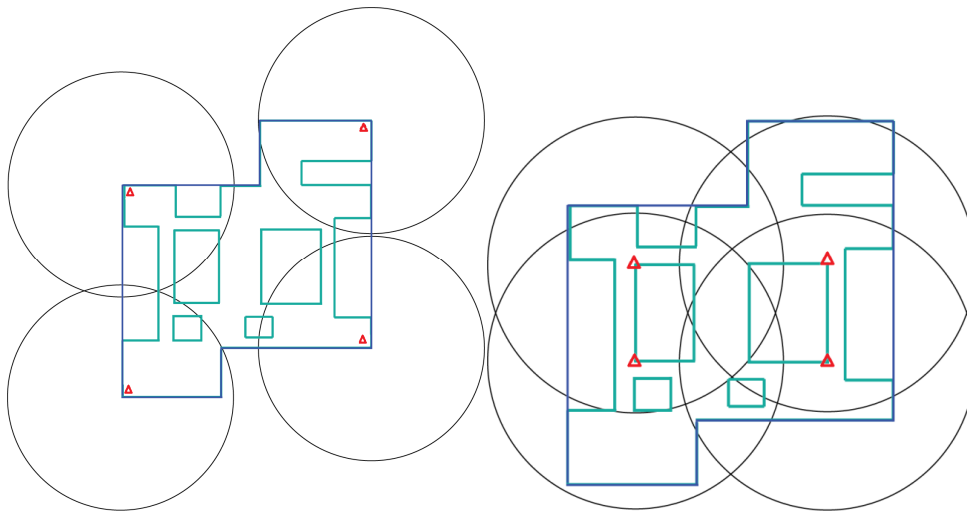
For subdividing the space in 'De Rotterdam', the Multi-Layered Space Model of IndoorGML is taken into consideration. Indoor space is represented by three different space layers which correspond to different space subdivisions:

- Functional layer, which mainly consists of meeting rooms, obstacles such as walls, and clearly defined spaces where the copy machines are for example. As well as the central blocks with the stairs, the elevator and the toilets in the middle (Figure 94 left).
- Navigational layer, which is composed by the corridor and the spaces where people use to walk through. The tables were not taken into consideration as obstacles, because of the accuracy of the localization methods (Figure 94 right).
- Range of the scanner layer, which is derived by the heat maps (Figure 95).

In IndoorGML, different decompositions result in different Node-Relation Graph (NRG), but in this case the team has decided to combine all this different layers and to extract only one network in the end.



*Fig.94. Functional layer (left) and Navigational layer (right)*

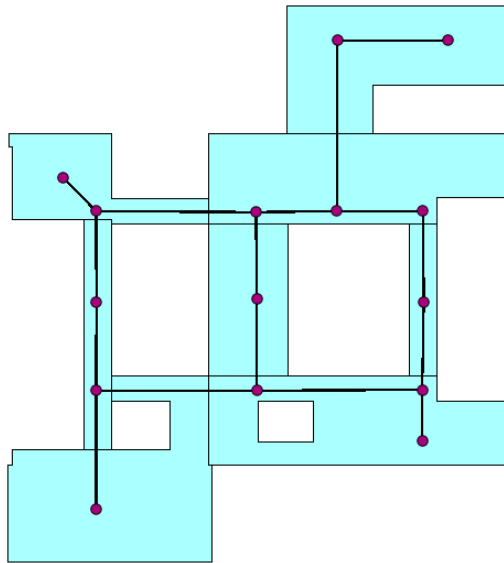


*Fig.95. Range of the scanner of 20 meters for layout 1 and 2.*

### 11.1.2 The manual network

If the network is drawn manually, for instance using a CAD software, it can be easily adapted to the characteristics of the building (obstacles, rooms, etc.), as in the case of the space subdivision. In order to automatically extract the network, the subspaces obtained with the space subdivision were utilized, except for the rooms (functional layer), which at first for the subdivision for the localization methods, were not considered. The functional layer is removed from the subdivisions, because it is considered to be an obstacle where you cannot walk through.

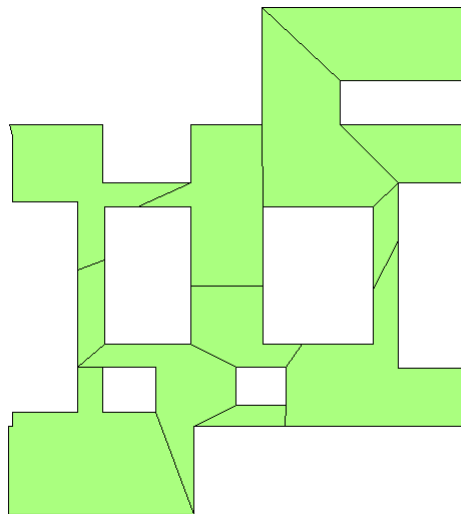
In the figure below (Figure 96) the manual network overlaid on the intuitive space subdivision can be seen. Basically each center of a workspace is connected to each other. The functional layer (the rooms etc.) describes a subspace. Since the localization is not accurate enough to localize inside a room, a connection to these rooms is not made. This is not necessary, because the localization methods give back one subspace, which can include just open space or a room and open space.



*Fig.96. Implementation of the manual derivation of the network on the intuitive subdivision*

### 11.1.3 Semi- automatic network

The subdivision used for testing an (semi-)automatic derivation of the network is the subdivision based on the range of the scanners, after removing the functional layer (Figure 97).



*Fig.97. Subdivision based on the range of the scanners, after removing the functional layer.*

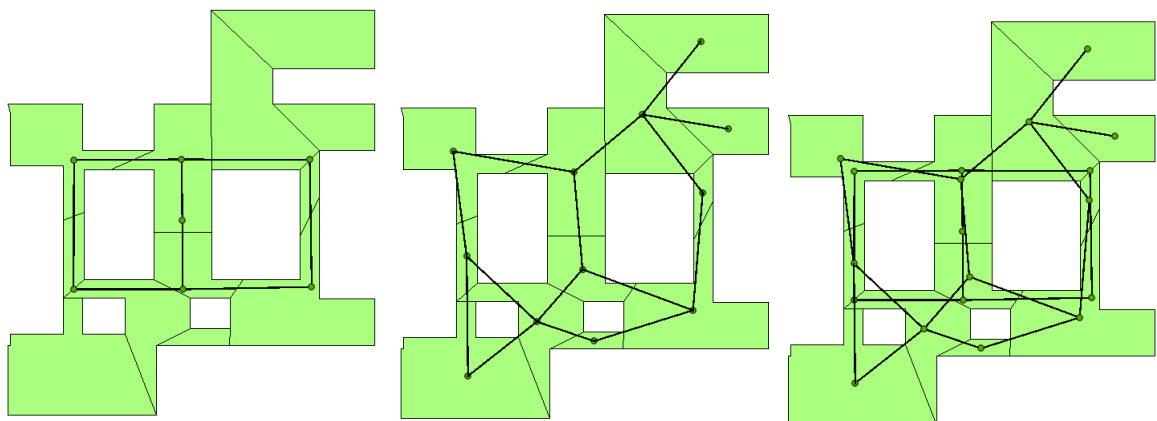
The semi- automatic method considers the building geometry. Certain nodes are necessary for an efficient routing for any floorplan in the MidTower of ‘De Rotterdam’ building, regardless of the space subdivision. For this reason, a ‘basic routing’ is created as a part of the network. It consists of nodes in and around the buildings core, enabling navigation from and to the elevators. Additional nodes for the routing around the core of the building are added to enable an effective routing. As this core is present throughout the entire tower, the ‘basic routing’ network is present at every floor. After this basic routing is in place, the subspaces resulting from the space subdivision can be taken into account. For each subspace the centroid was

computed using the Python library Shapely. The script then searches for the center points of the neighbouring subdivisions and connect these points with the route around the core (see Figure 98).



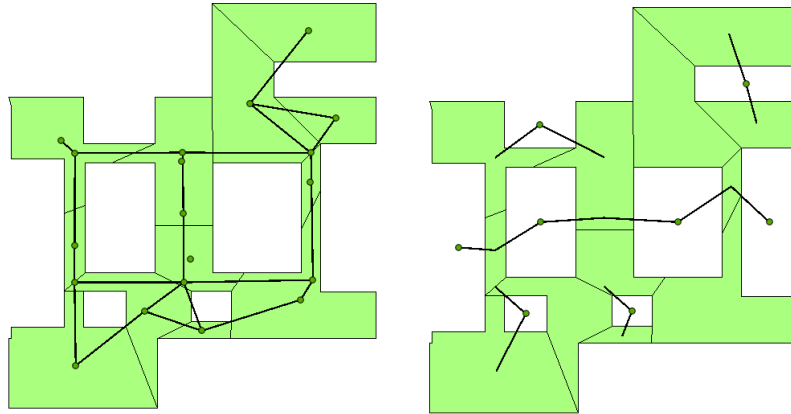
*Fig. 98. Concept for the semi- automatic derivation of the network*

The steps of the method can be seen in the figure below (Figure 99). The routing around the core can be seen on the left. In the middle are the center points of the subdivisions connected. The routing around the core is layered over the routing between the center points on the right.

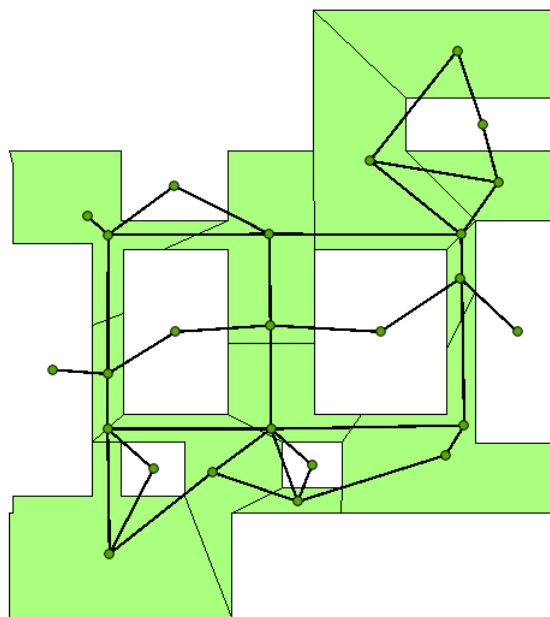


*Fig.99. Implementation of the semi- automatic method on the subdivision based on the range*

In the next steps the two routes are combined by choosing the shortest distance between the points. The crossing lines are removed on the left, in the figure below. For a future application, a connection to the rooms/obstacles is added. When then having a more accurate localization method, localizing inside a room will be possible. For the application, nodes need to be in the rooms, for the path finding algorithm to find a route to them. When the application will look for available workspaces, all the available workspaces need to be connected to each other. The connection to the holes can be seen in the figure below on the right. They are made manually, because the doors/openings need to be taken into consideration for the connection to the already there network. It could be possible to create a node for each door, and then automatically make the network, but that is not done for this project.



*Fig.100. Implementation of the semi- automatic method on the subdivision based on the range*

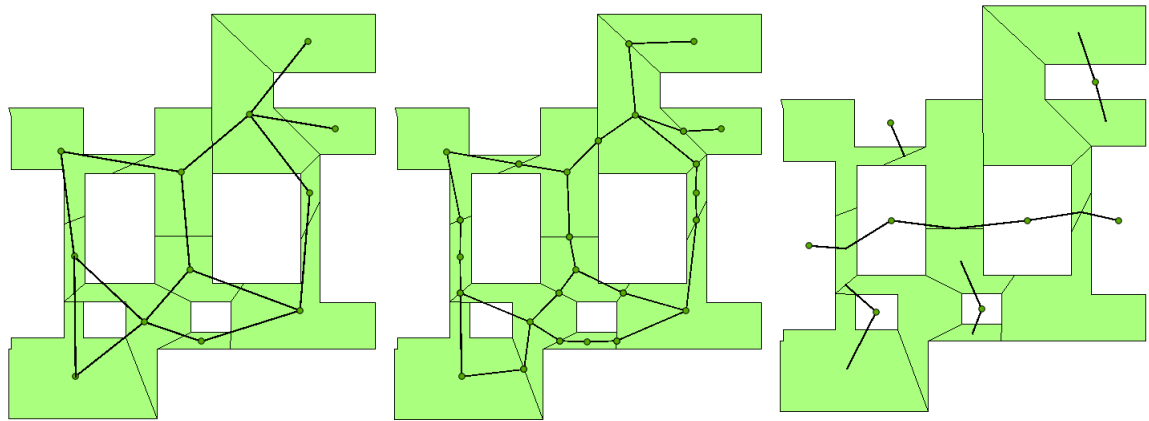


*Fig.101. The final network*

Looking at the figure above, it can be seen that the route is not perfect, because at certain places the route is cut short and crosses an obstacle, where you cannot walk through. The only connections allowed through the holes are the ones that are combined with the center of the holes and are going through a door or opening. Also there is a lot of manual editing needed, for example when the route around the core is combined with the route through the center points.

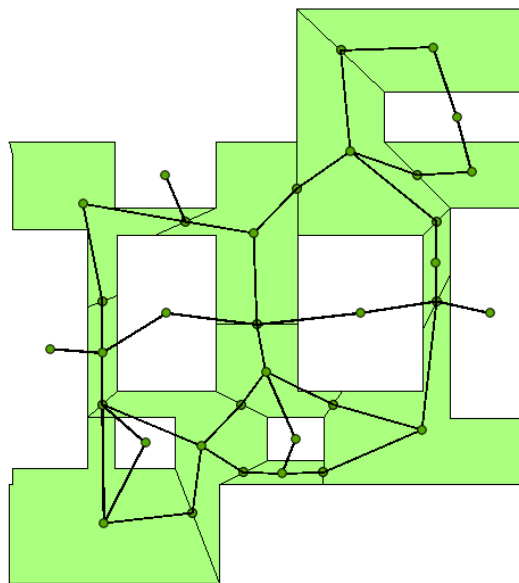
#### 11.1.4 Automatic network

The automatic network does not consider the building geometry. It uses the centerpoints of the subdivisions as a starting point. After computing the centroid, each centroid is connected with the centroids of the neighbouring subspaces and in this way the network has been generated.



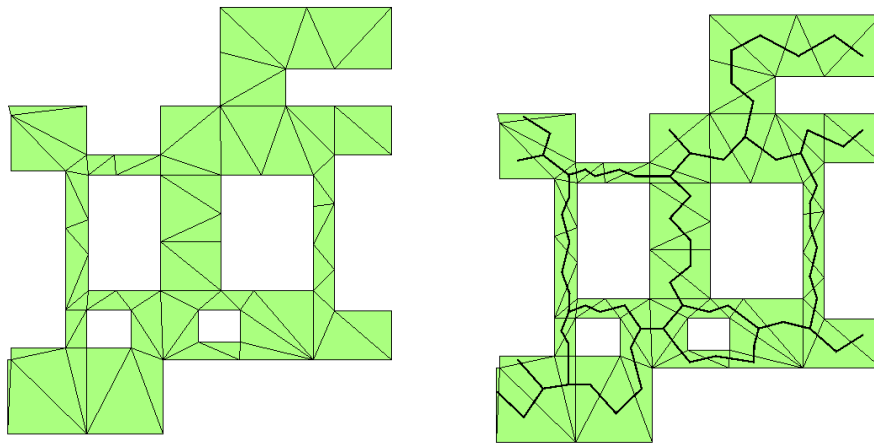
*Fig.102. Implementation of the automatic method on the subdivision based on the range*

As seen in the left figure above, the network crosses many holes. This is tried to solve by ‘guiding’ the lines through the middle points of the intersecting lines between the neighbouring subdivisions, as can be seen in the middle. Especially in the right upper corner, the route becomes a lot better. The connection to the rooms is made manually. Lines are made, considering the doors/openings of the rooms, to the closest point, as can be seen on the right. The final route can be seen in the figure below.



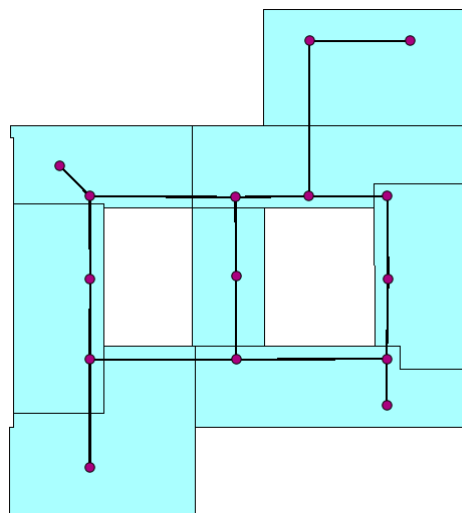
*Fig.103. The final network*

The final result seem better, in comparison to the semi- automatic method, better, because the network crosses fewer holes. Still it is not suitable for use, because there are still obstacles crossed. To solve this problem, another automatic derivation of the network could be drawn by using the centroids of the triangulated space and connecting them with each other. Now the network will not cross a hole. Access to the holes needs to be made manually, as done for the other networks. But it is not performed in the figure below, since the connection to the holes is not necessary, as explained before.



*Fig.104. Implementation of an automatic method on the triangulated floorspace*

A result can be seen above. The network contains a lot of nodes, which is not needed for the navigation, since the accuracy of the localization is not that precise. A solution to this problem could be to simplify the network. For example, every node that has three branches, should be kept, and the nodes in between removed. This will cause lines to cross the holes, so exception to this rule should be made. This method was not further investigated. By comparing all the different networks with each other, the manual network seems, for now, to be the most effective and will be used later on for the application.



*Fig.105. The network used for the application*

#### 11.1.5 Storing the network

Once the network is derived, the positions of the network nodes can be stored in a local coordinate system (x,y,z). Based on these positions, the lengths of edges can be calculated and the final network can be created. For the manual network in `De Rotterdam`, nodes are described as 4 digit integers, since the alphabet is not sufficient to describe all nodes. By using 4 digits the floor level can be indicated with the first 2 digits. The network is depicted in



the figure on the left below. The localization method gives back a nodenumber. This node is the representative node for that subspace, which can be seen in the figure on the right.

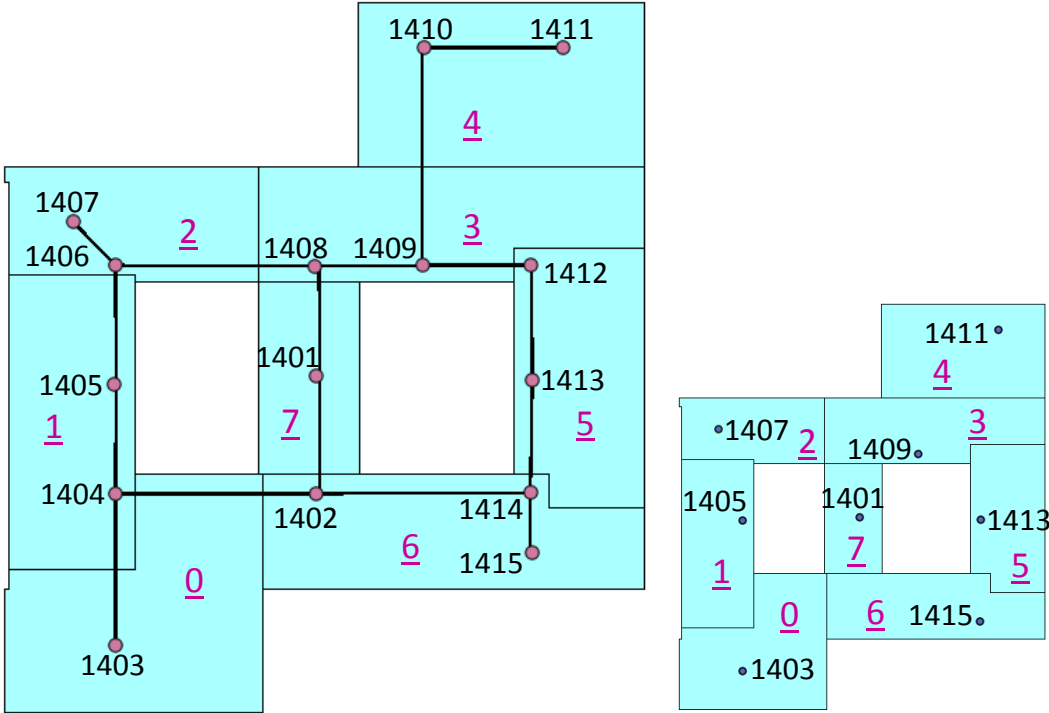


Fig. 106. Generated network with nodes

After retrieving all node positions and calculating the edge lengths, the generated network stored as an adjacency list looks as in Figure 107. In the following steps this network will be used in the path finding algorithm and in the visualization of the route navigation.

```
G =
1401 (1402, 9.1), (1403, 9.0), (1407, 10.9)
1402 (1401, 9.1), (1403, 9.0), (1407, 10.9)
1403 (1402, 9.0), (1404, 8.0)
1404 (1403, 8.0), (1405, 2.0), (1409, 10.9)
1405 (1404, 2.0)
1406 (1407, 2.0)
1407 (1402, 10.9), (1406, 2.0), (1408, 9.0)
1408 (1407, 9.0), (1409, 8.0)
```

Fig.107. Example adjacency list

11.1.6 Path finding

By using the A\* algorithm in complicated graphs, a lot of node visits can be avoided as the search is directed at the target node. However, in the current example network for de Rotterdam only few nodes are available.

If the searched colleague is located at node 1413, the users can be instructed to go to the 14th floor. It can be assumed that an employee knows where the elevators are, meaning that this part of the path finding is not necessary. Therefore, the path finding may start from the

`elevator-node` 1408 and create a route towards node 1413, regardless of the user's position. Instead of creating one large network for the whole tower or building, different network for different floor levels may be used. Target node 1413 is at the 14th floor, meaning that graph 14 can be searched through. Since there is only a small amount of nodes per floor level, the choice of a path finding algorithm will not have a significant effect on the computation times. Therefore the Dijkstra algorithm will be used, since it is the easiest algorithm to implement. If it is assumed that employees can use staircases to visit colleagues who are 1-2 floors away from them, the need of a network connecting all floor levels becomes greater. For these searches, a staircase routing may be computed. However, after visiting the building, the stairs are emergency exists and will not be taken into consideration as a route.

As the scanner data and thus localization will be processed on the database, the path finding algorithm will also run on the server side. This means that the algorithm can be run on most of the languages, and scripting this within the mobile phone application in Android is not needed. As most of the team is familiar with Python, for now a Python script will be used. A prewritten Dijkstra algorithm scripted in Python by David Eppstein (UC Irvine) was used.

Input is in the Adjacency list form, as described in the chapter Network. Where the network was previously described as:

```
G = 1401 (1402,9.1), (1403,9.0), (1407,10.9)
     1402 (1401,9.1), (1403,9.0), (1407,10.9)
     ...
```

the syntax needs to be adjusted to:

```
G = {'1401':{'1402':9.1,'1403':9.0, '1407':10.9'},
      '1402':{'1401':9.1,'1403':9.0,'1407':10.9},
      ...}
```

Running the algorithm with:

```
ShortestPath(G,1411,1405) with G = the name of the graph,
startnode = 1411 and endnode = 1405
results in ['1411', '1412', '1413', '1414', '1409', '1404', '1405']
```

This is to be expected, as it is the shortest path, computed by choosing the shortest edges (for example from node 1412 to node 1413 [weight 9.0] instead of node 1407 [weight=15.6]). The series of nodes, resulting from the Dijkstra algorithm, can now be used to navigate the user to the colleague.

### 11.1.7 ROUTE DESCRIPTION

Once the path from the user towards the colleague has been determined, the next step is to guide the user to the colleague. This can be done by a route description or a visualization of the route. In this section the possibilities of a route description in 'De Rotterdam' will be described.

One way to describe the route of a user is by explaining its path in text, relative to the observed environment. Examples of this are "*go left when leaving the elevator*" or "*at the main entrance, walk straight ahead*". As visible from the above statements, so called landmarks are needed to clearly describe a route. Directions are only meaningful when they can be related to distinguishable objects which the user can understand. Examples of landmarks could be stairs, elevators, corridors and junctions or other objects which can be identified in an unambiguous way.

Within ‘De Rotterdam’ some clear landmarks can be found. For example the elevator, doors and some corridors can be used. Possible route description statements and the related landmarks are shown in the figure below.

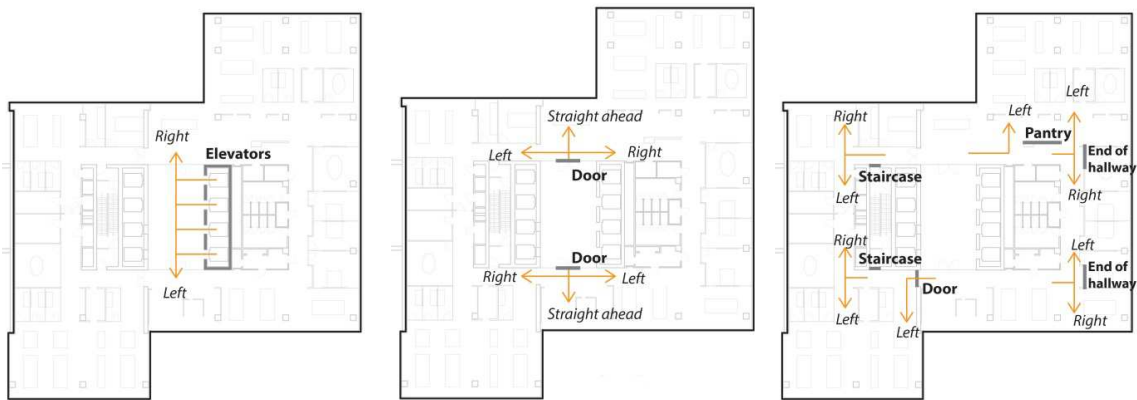


Fig. 108. Possible route description statements based on present landmarks

Once the correct floor plan is reached, one can leave the core of the building either through the left or the right. A statement could be “leave the elevator and go left”. Since the user needs to go through a door in order to exit the core, a second statement might be “go through the door and go right”. From this point onwards there do not seem to be any distinct landmarks available. Within the work floor, different workspaces are mixed thus no clear differences in furniture are present. A statement as “go right at the meeting room” may be interpreted wrongly if several meeting rooms are present. On one side of the building the staircases may be used as a landmark, although these are not clearly visible. On the other side, this is not the case, so describing the route relative to the corridor may be used, although this is not very clear. A statement could be “go right at the end of the hallway”. Some photos inside the building indicating the given statements are shown below.

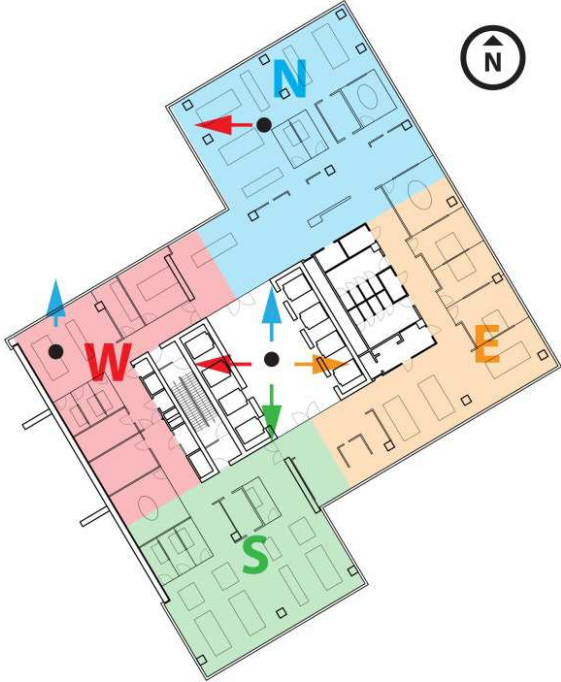


Fig. 109. Photos ‘de Rotterdam’, arrows indicate route description statements “left, right, right”

Other options are using room numbers or metric directions such as “go straight ahead for 10 meters and go left”, however these are difficult for the user to follow accurately. It can be concluded that a text based route description may be helpful to the users, but for certain areas there might be misunderstanding of the statements. To enhance the possibilities of navigating via these statements specific landmarks could be added or certain areas could be indicated

with materials/colours. However, assuming the building interior stays as it is, additional visualization of the route seems to be helpful to the user.

Another option is to use a route description combined with space subdivision. Subspaces may be defined as “north, east, south & west” where each cardinal direction describes a certain corner of the building. The mobile application may accompany the user with a compass, showing the orientation of the area where the target is situated. However, this option limits the space subdivision to 4 (n,w,s,e) or 8 (n,nw,w,sw,s,se,e,ne) areas. Also, this way of navigating may not function very intuitively as the direction of the arrow may point at a certain direction while the user is supposed to move towards somewhere else. As visible in figure 40 a user being navigated from the elevator can be guided based on the orientation. However, if the user is not positioned in the middle of the floorplan, the cardinal direction might be confusing or hard to use.



*Fig. 110. Navigation based on cardinal direction*

## 12. Visualization

In order to navigate an employee towards a colleague, visualization might be the most direct way of achieving this. Also, selecting the colleague that is being searched should be as easy as possible. As the employees of the Rotterdam municipality are granted mobile phones which run on Android, the application has to be developed for this platform. There is no need to consider iPhones or other mobile platforms. During this project, Eclipse was used for the android application development. This chapter will start with a description of the general functionality of the application. This will be followed by a more detailed description of the separate components. Pieces of code will be given to illustrate the most important parts of the process. Larger parts of the code can be found in the appendix.

### 12.1. APPLICATION FUNCTIONALITY

As mentioned previously, the main aim of the application is to navigate a user to another colleague. Since, there are around 3000 employees working for the municipality of Rotterdam, all of whom can be present in the office, it should be easy to select the colleague one is looking for. The first step is to acquire the mac-addresses of the searching and the searched colleague. To select the searched colleague, the relevant department can be chosen out of a list of departments. The following step is to select a colleague that belongs to the selected department (see Figure 111).

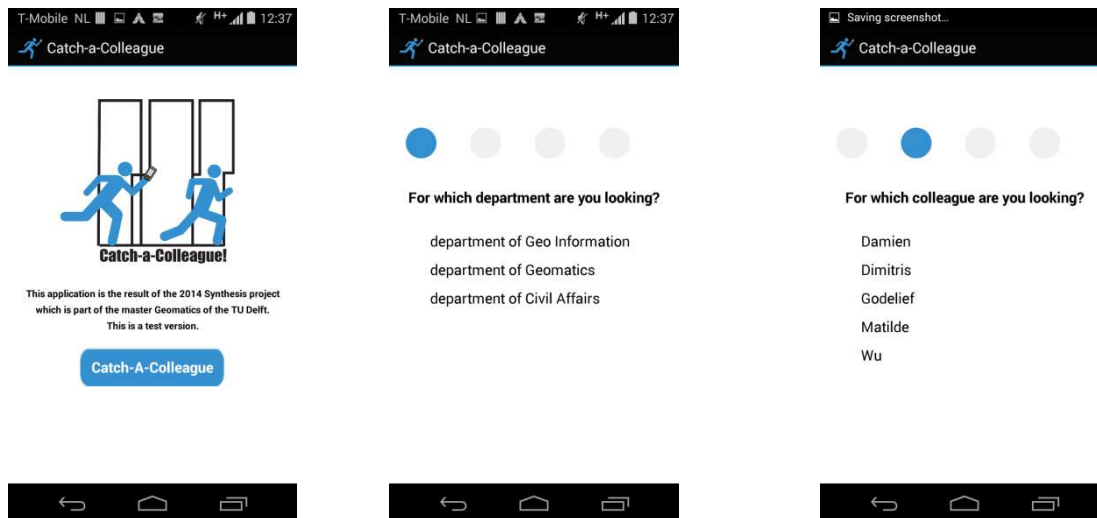


Fig. 111. Main screen, department selection and colleague selection in mobile application

A connection to the database has to be made in order to retrieve the mac-address of the searched colleague. Frequently updating and validating this database with the correct mac-addresses for each colleague is crucial for the functionality of the system. Since the users' location is also required, its mac-address should also be retrieved. This can be done within the mobile device itself. Once these values are found the localization script can be requested to run with the two mac-addresses as input nodes to retrieve the shortest route (see Figure 112).

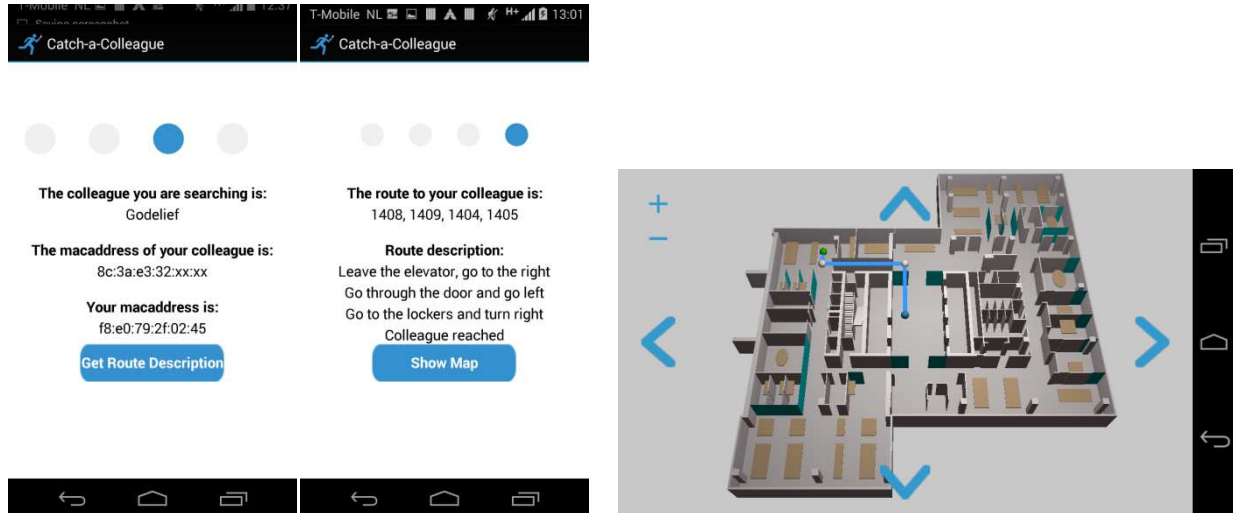


Fig.112. Selection overview and route description screen in mobile application.

These scripts are working on the server side. Another table can be queried for all node pairs of the route, in order to retrieve the route description. While launching the map view the route is sent to the Unity application, so it can be rendered on the screen.

## 12.2. DATABASE CONNECTION

To be able to store mac-addresses of all colleagues, a database is needed. Two tables are created, one table storing data about departments and another table for individual phones. The following tables have been created via the MySQL workbench.

depid	department	Created_at	Updated_at
1	department of Geo Information	2014-10-21 13:25:55	0000-00-00 00:00:00
2	department of Geomatics	2014-10-21 13:25:55	0000-00-00 00:00:00
3	department of Civil Affairs	2014-10-21 13:25:55	0000-00-00 00:00:00

Table 14. Departments table in MySQL database

phone id	name	macaddress	department	depid	Created_at	Updated_at
1	Rob Poll	d3:5b:32:d2:xx:xx	department of Geo Information	1	2014-10-21 13:29:51	0000-00-00 00:00:00
2	GeoInfo Colleague 1	d3:5b:32:2a:xx:xx	department of Geo Information	1	2014-10-21 13:29:51	0000-00-00 00:00:00
..	..	..	..	..	..	..
5	Damien	0c:14:20:6e:xx:xx	department of Geomatics	2	2014-10-21 13:29:51	0000-00-00 00:00:00
6	Dimitris	d4:32:a2:41:xx:xx	department of Geomatics	2	2014-10-21 13:29:51	0000-00-00 00:00:00
..	..	..	..	..	..	..
10	Civil Affairs Colleague 1	f8:e0:79:2f:xx:xx	department of Civil Affairs	3	2014-10-21 13:29:51	0000-00-00 00:00:00
11	Civil Affairs Colleague 2	f8:e0:79:c1:xx:xx	department of Civil Affairs	3	2014-10-21 13:29:51	0000-00-00 00:00:00
..	..	..	..	..	..	..

Table 15. Phones table in MySQL database

In order to query these tables from the application PHP is used. The PHP-file is stored on an external server. The URL to this PHP-file is stored and a GET-request is performed on this URL. A list of JSON strings is returned and then read by a JSONParser. This is exemplified in the following code lines (AllPhonesActivity.java).

```
private static String url_all_phones = "http://server.kirupa.nl/damien/get_department_details.php";  
    JSONObject json = JSONParser.makeHttpRequest(url_all_phones, "GET", params);
```

The php-file queries the database and reads the values into a list. This list is then returned to the GET request. (get\_department\_details.php, code snippet)

```
$result = mysql_query("SELECT * FROM catchphones WHERE depid = $depid");
```

In the above example, the query for colleague selection is given. In a similar way but with a different query, all departments are shown. After the colleague selection, its phoneid, name and mac-address are passed on to the next activity (AllPhonesActivity.java).

```
Bundle extras = new Bundle();  
extras.putString("TAG_PHONEID", phoneid);  
extras.putString("TAG_NAME", name);  
extras.putString("TAG_MACADDRESS", macaddress);  
Intent in = new Intent(getApplicationContext(), SelectPhoneActivity.class);  
in.putExtras(extras);
```

Retrieving the mac-address of the phone of the users of course doesn't need a database connection. It can be acquired with the following code (SelectPhoneActivity.java).

```
Bundle extras = getIntent().getExtras();  
String macaddress = extras.getString("TAG_MACADDRESS");  
WifiManager manager = (WifiManager) getSystemService(Context.WIFI_SERVICE);  
WifiInfo info = manager.getConnectionInfo();  
String address = info.getMacAddress();
```

This results in the colleague selection screen, displaying the mac-adresses of both the user and searched colleague.

### 12.3. PYTHON LOCALIZATION

Since the localization algorithm was written in Python, a way has to be found to run the code with the input from the application. One way of doing so is using a Scripting Layer for Android (LS4A). However, a web service is the preferred option as described previously in the system architecture. In a similar fashion as the previous step, a PHP file is called which calls a python script on a remote server, to run with two mac-adresses as input.

```
(java) private static String url_run_python =  
    "http://server.kirupa.nl/damien/get_python_route.php";  
(PHP) $myroute = system('python myscript.py mac1 mac2', $retval);
```

The python code consists of the localization algorithm which returns a location as a network node. Running a shortest-path algorithm on these two nodes will result in a route given as a sequence of node numbers. When the searching user is localized on a different floor level than the searched colleague, the shortest path start node may be assigned to the 'elevator-node'. At this point the user will be instructed to go to the right floor and a route description in language can be given.

## 12.4. ROUTE DESCRIPTION

In the application a sample route description is shown. By creating a database with route description for each node pair, this description could be automatically gathered from just the node sequence. This method is easy to set up but does not take into account the direction in which a user is walking. The following table shows the idea.

Routeid	Startnode	Endnode	Route description
1	1408	1409	Leave the elevator and go right
2	1408	1407	Leave the elevator and go left
3	1409	1404	Go through the door and go left
4	1409	1414	Go through the door and go right
5	1407	1402	Go through the door and go right
6	1407	1412	Go through the door and go left
7	1404	1405	Go straight ahead until the lockers and go right
8	1404	1403	Go straight ahead until the lockers and go left
..	..	..	..

*Table 16. Route description based on node-pairs*

This method based on node-pairs is only valid when the start node is at the elevator node. When the searching colleague is at the same floor level a node sequence of 1414 → 1409 → 1404 might be possible. In this case the statement “Go through the door and go left” is not correct. Taking searching within the same floor level into account for the route description means that node-triples can be used to create statements which are correct in any case. This idea is shown in the following table.

Routeid	Startnode	Middlenode	Endnode	Route description
1	1408	1409	1404	Leave the elevator and go right Go through the door and go left
2	1408	1409	1414	Leave the elevator and go right Go through the door and go right
3	1414	1409	1404	Go straight ahead until the lockers
4	1414	1409	1410	Go straight ahead and go right and the doors
5	1409	1404	1405	Go straight ahead until the lockers and go right
6	1409	1404	1403	Go straight ahead until the lockers and go left
..	..	..	..	..

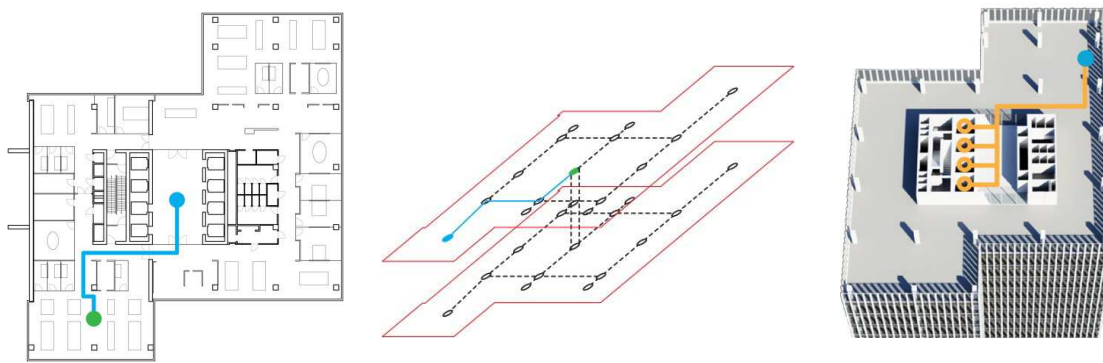
*Table 17. Route description based on node-triples*

This method is always correct because it takes into account the direction of the user, also in the case of same floor level searches. However, more statements need to be stored and retrieving the right statements takes some more computation. This part is not implemented in the application as it is considered of less importance and outside the scope of the project.

## 12.5. UNITY VISUALIZATION

Once the route is known a visualisation which can aid the user with its navigation can be created. To do so, the route should be rendered and accompanied by an illustration of relevant building parts. This visualisation can be done in 2D, 2.5D or 3D, depending on the demands and the complexity of the building geometry.





*Fig.113. 2D, 2.5D and 3D visualisation of routing within 'de Rotterdam'*

In the examples above (Figure 113), it is visible that routes can be displayed in each of these options. For a visualisation in 2D, only parts of one floor plan can be shown at a time. This means that complicated geometry such as height differences or split levels cannot be displayed. However, it does depict a clear route and it is easy to run on a mobile application.

The 2.5D visualization can show more complex routes, also in between levels. Height differences can be shown and several floors can be visualized simultaneously in a clear way. However, it is quite abstract and not a lot of information about the building geometry is shown. Therefore the user might have a hard time, understanding the showed route.

The most extensive option is to make a 3D visualization, depicting (parts of) a 3D-model on the mobile phone, while rendering nodes and lines on top of this. Showing or switching between different floor levels (visible buildings parts) and levels of detail (layers) belongs to the possibilities. More information about the building geometry can be shown, which may help the user. However, it could be heavy to run as mobile application.

Some of the floor plans of 'de Rotterdam' are complicated; there is for instance a split level with double height. Because of this reason, and along with the educational aspect, a 3D visualization will be made.

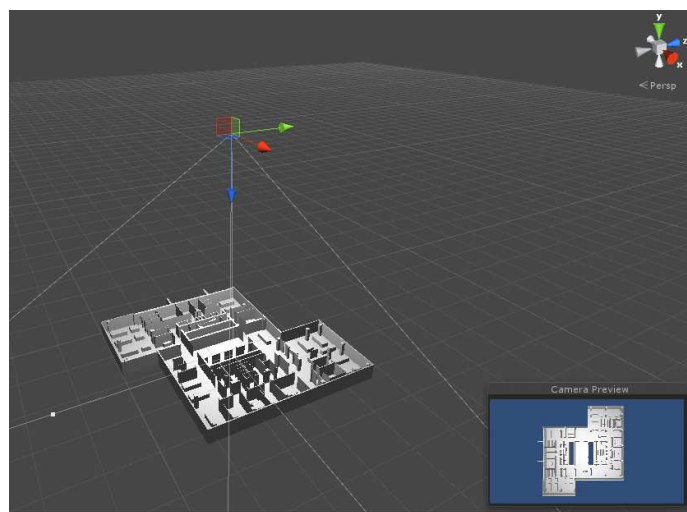
To render the geometry and routing, a 3D engine needs to be used. Based on last year's synthesis project, Unity3D is known to be capable of visualizing the geometry and routing. More importantly, it is compatible with Android applications and plenty of documentation is present. Within Unity3D a scene is to be created, consisting of a 3Dmodel, lighting, textures, camera control and a rendered route. The implementation of these aspects will be explained below.

### 12.5.1 Geometry

The municipality of Rotterdam provided the floor plans of the middle tower in PDF format. Out of these files the geometry of the facade, walls, columns, doors and furniture could be extracted. This 2D information was saved as .dxf and imported in Rhino. The 3D-model was manually created from this point on by turning the polylines into planar surfaces, which can then be extruded into solids. Some simple shaders were used to enhance the model. Geometry was created for the 14<sup>th</sup>, 15<sup>th</sup> and 16<sup>th</sup> floor of de Rotterdam.

There are several options for storing the geometry, but 2 were mostly considered. These are:

- Store all geometry on the server, where the required parts can be queried by the application on run time.  
The advantage of this is that the application will have a much smaller file size. However, more data transfer will be needed between server and application. In case of a full building model, this would lead to excessive file sizes. However, the model created for the three floor plans is not very rich in semantics nor is it geometrically detailed.
- A second option is local storing of the geometry on the phone. To visualize the route clearly, only the floor plan, on which the searched colleague is present is needed. Simple shaders are used to show the difference between solid walls and glass panels and furniture and floor. A point light above the geometry is used for illumination. This was finally the preferred method.



*Fig.114. Overview of the unity scene*

The geometry is loaded based on the final node, nodeIDs are given based on the floor level. Therefore first node of the 14<sup>th</sup> floor is node 1401 (routeRenderer.js code snippets).

```
var finalFloor : int = (finalNode/100);
  for (var geomIndex : int = 14; geomIndex < 17; geomIndex++)
    {if (finalFloor != geomIndex)
      {var tempGeom : GameObject = GameObject.Find("RdamFloor"
        + geomIndex); tempGeom.SetActive (false);}}
```

If the final node is 1405, the node is present on the 14<sup>th</sup> floor (final floor). All geometry is looped to check if the name matches RdamFloor14. If this is not true, the geometry is set to not active.

### 12.5.2 Camera control

The camera determines which part of the geometry is visible and in what way. In our case a bird's eye view perspective is chosen as it created the most oversight of the building. Although the entire floor plan is visible from the initial view, users may wish to change the

camera position in order to get more information of the building geometry. This camera control can be implemented in different ways:

- A good solution could have been touch-screen swiping and zooming. However, when this was implemented it resulted in unexpected behaviour as the camera was often shaking for a non-moving touch.
- Another option is using buttons as joysticks, panning and rotating the model by pressing a certain part of the joystick. Although this may be a functional option it doesn't seem very intuitive to use and requires some explanation for the user.
- Finally it was decided to use simple zooming and panning buttons, on each click the camera's x, y or z position is altered by a single step size. The step size was made dependant on the camera height, meaning that a zoomed out camera will be moving more distance with each click. This way, not many clicks are needed to get back to a more detailed view.

This last option can be understood by the following code (rightButtonScript.js)

```
var currentX : float = Camera.main.transform.position.x;
var currentY :float = Camera.main.transform.position.y;
var currentZ :float = Camera.main.transform.position.z;
var stepSize : float = 30/currentY;
Camera.main.transform.position = Vector3(currentX+stepSize, currentY, currentZ);
```

Similar code is used in all zoom and pan buttons to create a simple way to navigate through the geometry. A better implementation might be to zoom the camera to a certain node, and using the left/right/up/down buttons to move along edges to other nodes. This method, however, only makes sense in an orthogonal network.

### 12.5.3 Route renderer

The aim of the route renderer is to display the route on the screen. A route array may look like [1408, 1409, 1405, 1404] meaning the starting node is 1408 and colleague is located at node 1404. The first step is to loop through the array and to retrieve the node positions from a text-file. This file may be stored on a database, to enable updates on the network. In this version, however, the node positions are stored locally on the phone. Relative to the geometry the file-size of a text-file is not significant. The node positions table is shown on the right. Horizontal positions are indicated by the x- and z values, vertical positions are set above the floor plan with y-values of 4 (see table 18).

NodeID	x	y	z
1401	08.3	4.0	04.0
1402	07.6	4.0	13.0
1403	07.6	4.0	21.0
1404	07.6	4.0	29.0
1405	07.6	4.0	31.0
1406	18.5	4.0	11.0
1407	18.5	4.0	13.0
1408	18.5	4.0	21.0
1409	18.5	4.0	29.0
1410	18.5	4.0	31.0
1411	34.0	4.0	11.0
1412	34.0	4.0	13.0
1413	34.0	4.0	21.0
1414	34.0	4.0	9.0
1415	34.0	4.0	36.0

Table 18. Node positions floor

Looping through all route nodes returns a route position array, in this case:  
[[18.5,4.0,21.0],[18.5,4.0,31.0],[07.6,4.0,29.0],[07.6,4.0,31.0]]

Then this array is split up into pairs, resulting in the start- and end node of individual line segments.

```

Pair1 = [18.5,4.0,21.0] → [18.5,4.0,31.0]
Pair2 = [18.5,4.0,31.0] → [07.6,4.0,29.0]
Pair3 = [07.6,4.0,29.0] → [07.6,4.0,31.0]

```

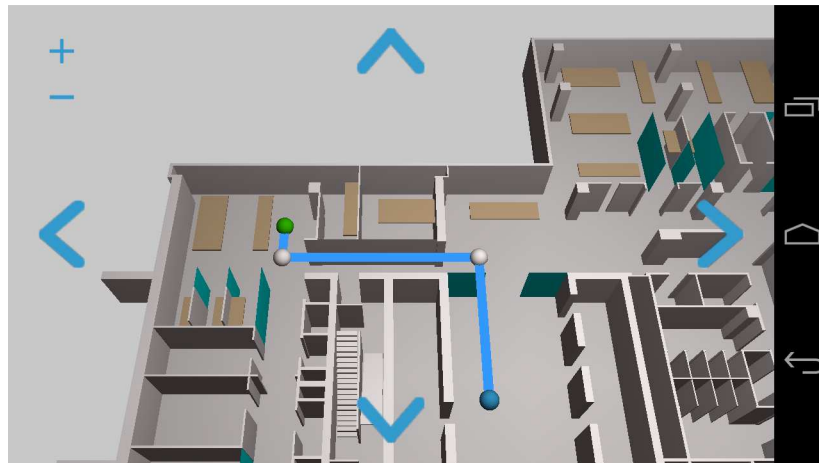
For each of the line segments a new `GameObject` is made to which a line renderer is added. Then the start and end node are set for each line segment (`routeRenderer.js`).

```

var pairStart : Vector3 = routeVectorArray[index];
var pairEnd : Vector3 = routeVectorArray[index+1];
var tempObject : GameObject = new GameObject();
var renderer : LineRenderer = tempObject.AddComponent(LineRenderer);
renderer.SetVertexCount(2);
renderer.SetPosition(0, pairStart);
renderer.SetPosition(1, pairEnd);

```

In a similar way all nodes are also looped through and spheres are placed on their positions. The starting node is set to blue, the middle nodes are white and the final node is green. Further guidance could be given as text statements in the screen or highlighting of the subdivision of the colleague's location. However, since employees will be familiar with the indoor environment this simple route is considered to be sufficient for navigation purposes.



*Fig.115. Table of node positions on floor 14*

#### 12.5.4 Integration of Unity within Android

In order to launch Unity from within the android application, some integration is necessary. The Unity project can be exported as an android project in several ways. This project can then be used as a library for the application. The android application activity may then extend on the `UnityPlayerNativeActivity` library as shown below. The route array which is received in the Android application can be communicated through to Unity. To achieve this, the name of the game object, the variable name and the actual array are given (`MapViewActivity.java`).

```

public class MapViewActivity extends UnityPlayerNativeActivity {
    ...
    UnityPlayer.UnitySendMessage("Point light", "routeArray", "[1408,
    1409, 1405, 1404]");
}

```

Currently the Unity application is functioning on its own within Eclipse. However, when activity extends on the UnityPlayerNativeActivity, this results in a OpenGL called out of context error. Until the final version of the report, efforts will be made to solve this problem.

## 12.6. CONSIDERATIONS

Simple solutions were developed in order to show the route and move through the geometry. Since the floor plan fits on a phone screen in a large enough scale, these solutions were adequate to communicate the route to the user relative to the building geometry. In larger or more complicated building parts however, more sophisticated control for the camera may be required for an user friendly application.

Based on the accuracy of the localization method, showing a single node as final destination might be misleading. As the localization method points towards a certain subdivision of the building, a good option may be to integrate the subdivisions into the building geometry. This way, parts of the floor could be highlighted by colouring or illumination, illustrating the expected presence of the searched colleague. Since the localization methods have an acceptable accuracy when the first and second subdivision estimates are combined, showing both of these areas in the application makes sense. However, as the research on the subdivision was performed simultaneously with the visualisation development there was no time to further research this option during this project.

Options for a route description based on node-pairs or triples have been discussed. Since most users will be familiar with the building, fast route visualization will be most effective in showing the colleagues location. For indoor environments with complex spaces route descriptions may be of benefit to the user.

Considering the concept system architecture, as seen in figure 10, a few changes have been made for the real implementation. In the database, the scanner data is not real time dynamic data, but data gathered after two days of testing in ‘De Rotterdam’ building. The connectivity data, which contains the network, is not stored as a table in the database. The geometry, which contains the manual made 3D model, is stored locally on the smartphone.

<b>Component</b>	<b>Implementation</b>
<b>STORAGE: external server</b>	
SQL Database	MySQL Workbench stored on external server
Scanner data	Not real-time data
Employee data	Department & colleague table, MAC-addresses linked to colleagues
<b>MIDDLE WARE: Web service on external server</b>	
Location algorithm	Fingerprinting & Multi-lateration calling the database
Path finding algorithm	Dijkstra
<b>THIN CLIENT: Locally stored</b>	
3D rendering	Unity3D
Smartphone application	Android application running PHP-files
Geometry	Geometry stored on smartphone

*Table 18. Implementations of the components in the prototype*

In general it can be concluded that the challenges of creating an indoor navigation application do not lie in the loose components. The integration of the system requires both some technical knowledge and resources. Communicating between different parts of the system is crucial to create an user friendly application. However, developing the layout and visualization should not solely focus on user friendliness but also represent the underlying process so that the outcome is not misleading to the user.

## Conclusions

The project at hand tries to achieve a complete solution to the problem faced by the Municipality of Rotterdam. The motivation for the created application was for an employee to be able to find the colleagues in the massive building of 'De Rotterdam'. The application consists of a localization, a navigation and a visualization component. The research question focuses mainly on the localization component, but all these components are connected to each other and influence the question.

The research question was: To which extent is it possible to localize a colleague using Wi-Fi Monitoring on a single floor level in 'De Rotterdam' building?

This relates to the space subdivision, scanner placement, localization and the extraction of a navigable network. They interact with each other and one general solution cannot be drawn. The way each of them is defined determines the direction the other one should follow in the search for a best outcome.

The greatest impeding factor in achieving a better to perfect result for the localization, is that the physical world environment interacts with the signals received by the scanners. The physical environment, or building geometry, has influence on the scanner range. For example, the concrete building core blocks the signal, multipath and noise can arise due to the materials of furniture, walls and other attenuation factors. This leads to the decision: you either need to model the environment, or neglect it. This also relates to the scanner placement and the number of scanners used. If you choose to not model the environment, the scanner placement becomes much more important. The scanners' coverage area but also the discreteness of the areas they each cover may dictate the outcome of the data collection.

The space subdivision is dependent on the accuracy of the localization method that the application tries to achieve. At the same time, the number of subspaces and the way the space is divided has an effect on the localization methods.

The data collected for the localization methods consisted only of the RSSI values received by the scanners. These values can be ambiguous as far as relating them to distance, but can be still used as indicators. The multilateration method described tries to exploit this fact and to some extent it succeeds. For the multilateration the results are that 6 out of 10 times a device is correctly localized within a subdivision and that result rises to 9 out of 10 times success, when the next possible subdivision is included in the solution.

Wi-Fi fingerprinting has a better performance in indoor localization since it does not use any mathematical method to model the physical environment. Instead, it collects samples and uses them as a-priori knowledge to derive a solution. However, the heat maps it creates highly rely on the specific set-up, from the placement of the scanners to the interiors indoor. Every time an element of the set-up changes, the data need to be recollected to update the heat maps, which can be very labor-intensive and time-consuming. The results of the fingerprinting are that 6 out of 7 times a colleague is localized correctly.

Related the results to the research question, the extent to which localization can be achieved varies depending on the localization method chosen but also the subdivision precision to be expressed to the user. Given a subdivision that logically segments the physical environment into neither too many nor too few subdivisions, is displayed as a second option. Wi-Fi fingerprinting is more promising because it can reach up to 100% success rate, given a better testing phase is performed beforehand.

With respect to the space subdivision, two different space subdivisions have been implemented: the intuitive space subdivision and the automatic space subdivision. In the end, the intuitive space subdivision seems to be the most suitable for being used to navigate the

user in the building, since it is based on the usage of space. Therefore, the intuitive space subdivision has proven to be more human-understandable because the different subspaces are created in a way that is comprehensible and logical for humans to understand. In fact, once the user is navigated to a certain node in the building, he/she can easily look around in the subspace, where the node is located, to try to find his/her colleague.

Within this project, eight different subspaces were considered to be a fair compromise, considering the localization which is not accurate enough to localize a person inside a room. With a much more accurate localization, the subdivision could be more detailed.

For the navigation component, in total three networks have been implemented: the manual network, the semi-automatic network and the automatic network. By comparing all the different networks with each other, in the end the manual network seems to be the most effective since it can be easily adapted to the characteristics of the building. In fact, the manual network has been drawn manually considering the navigable spaces, where people can walk through, the rooms and the obstacles. In the end, the network is rather coarse, which relates to the localization accuracy. Since the building mainly consist of open spaces and glass walls, the user, once navigated to a certain node, can easily look for his/her colleague. Further investigation might be performed in order to automatically achieve a network that does not cross obstacles and does not contain too many nodes which are not needed for the navigation.

With respect to the application, it can be concluded that for creating an user friendly and reliable indoor navigation application, the integration between the different components is crucial.



## Recommendations

For further research, a few considerations can be taken into account. For example the indoor localization, by using four Meshlium scanners, may be improved by:

- In depth research on scanner hardware  
With more knowledge about the hardware used, different scanner layouts can be taken into consideration
- Gathering more testing data  
By doing more tests, more data can be gathered which can be used for the Wi-Fi fingerprinting. With more data, better heatmaps can be created and the localization will be more reliable.
- Adjusting scanner layout for a specific localization method  
The scanner layout relates to a localization method.
- Adjusting space subdivision based on method, layout and application  
The space subdivision can increase the localization accuracy.
- Testing floor level separation  
Especially for 'De Rotterdam' building with its many floors, testing floor level separation is important, since colleagues can work on different floors. It is important to know if the floors will block the signal of the Meshlium scanners.

## References

- Afyouni, I., Ray, C. and Claramunt, C., 2012. Spatial models for context-aware indoor navigation systems: A survey. *Journal of Spatial Information Science*, 4, pp. 85–123.
- Bettini, C., Jajodia, S., Samarati, P., Wang, S.X. (2009) Privacy in location-based applications: research issues and emerging trends (Lecture Notes in Computer Science / Information Systems and Applications, incl. Internet/Web, and HCI), Springer, 2009 edition (September 10, 2009) ISBN 978-3-642-03511-1
- Biswas, J. and Veloso, M., (2010). WiFi Localization and Navigation for Autonomous Indoor Mobile Robots, Carnegie Mellon University, Pittsburgh, 2010
- Ching, W., Teh, R.J., Li, B., and Rizos, C., (2010). Uniwide WiFi Based Positioning system, University of New South Wales, 2010
- FIDIS (Future of Identity in the Information Society), 2009. The legal framework for location-based services in Europe.  
<http://www.fidis.net/resources/fidis-deliverables/mobility-and-identity/d115-the-legal-framework-for-location-based-services-in-europe/doc/42/>
- Jenkins P, Phillips T, Mulberg E and Hui S, 1992, Activity patterns of Californians: Use of and proximity to indoor pollutant sources. *Atmospheric Environment - Part A General Topics*, 26A(12): 2141-2148, 1992
- Liu, K., Liu, X., and Li, X., (2013). Guoguo: Enabling Fine-grained Indoor Localization via Smartphone. *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pp. 235-248, New York, 2013
- Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, and B., Konolige, K., (2010) The Office Marathon: Robust Navigation in an Indoor Office Environment, Willow Garage Inc., USA, 2010
- Meijers, M. Zlatanova, S., and Preifer, N. (2005), 3d geoinformation indoors: structuring for evacuation, in *Proceeding of Next generation 3D city models*, 21-22 June, Bonn, Germany, 2005
- Navizon: <http://www.navizon.com/its/whitepaper.pdf>
- Paul, A.S., and Wan, E.A. (2008). Wi-Fi based indoor localization and tracking using sigma-point kalman filtering methods. *OGI school of Science and Engineering*, 2008
- Quintas, J., Cunha, A., Serra, P., Pereira, A., Marques, B., and Dias, J. (2013). Indoor Localization and Tracking Using 802.11 Networks and Smartphones. In *Evaluating AAL Systems Through Competitive Benchmarking Communications in Computer and Information Science*, Volume 386, pp 117-127, 2013
- Radu, V., Kriara, L., and Marina, M.K., (2013). Pazl: A Mobile Crowdsensing based Indoor WiFi Monitoring System, University of Edimburgh, 2013
- Schiller, J., and Voiserd, A., (2004). *Location-Based Services*. Elsevier, San Francisco, 2004
- Telecommunicatiewet.  
[http://www.government.nl/documents\\_andpublications/notes\\_/2012/06/07/dutch-telecommunications-act.html](http://www.government.nl/documents_andpublications/notes_/2012/06/07/dutch-telecommunications-act.html)

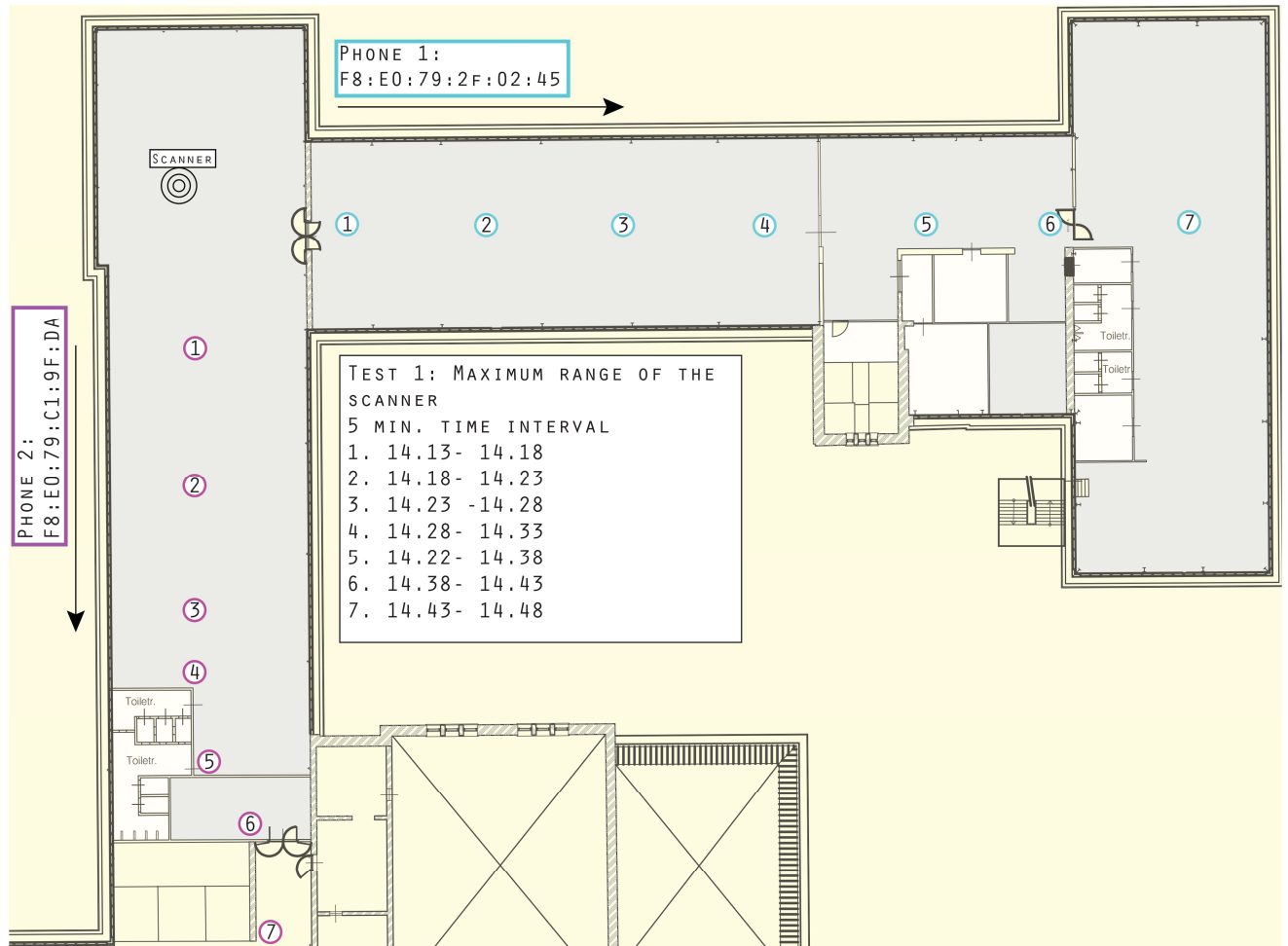
- van Loenen, B., Zevenbergen, J., De Jong, J. (2008) Geo-information: what is it and what is the legal context? L van Wees & S Nouwt (Eds.)
- van Ooijen, C., Nouwt, S. (2009) Power and Privacy: the Use of LBS in Dutch Public Administration. Nederlandse Commissie voor Geodesie Netherlands Geodetic Commission 48, 2009.
- Wet bescherming persoonsgegevens. <http://www.ivir.nl/wetten/nl/wbp.pdf>
- Wet bescherming persoonsgegevens. <http://www.cbpweb.nl/Pages/home.aspx>
- Worboys and Duckham, *GIS: A Computing Perspective*, Second Edition, 2004, CRC Press
- Zlatanova, S., Liu, L., Sithole, G., Zhao, J. and Mortari, F., 2014. Space subdivision for indoor application. GIST Report No. 66, ISBN: 978-90-77029-37-4
- Zomax Wireless, “RCP Installation Considerations and Principles”, Zomax Technologies Inc, 2010  
[http://www.zcomax.com/rcp/Rural\\_Connectivity\\_Platform-Installation\\_Considerations.pdf](http://www.zcomax.com/rcp/Rural_Connectivity_Platform-Installation_Considerations.pdf)
- J.Mulligan, “A Performance Analysis of a CSMA Multihop Packet Radio Network”, Chapter 7: “Indoor Radio Propagation”, Faculty of Virginia Polytechnic Institute and State University, 1997.  
<http://scholar.lib.vt.edu/theses/available/etd-51997-22830/unrestricted/Ch7.pdf>
- O.Tekdas, V.Isler, “Sensor Placement for Triangulation Based Localization”, IEEE Tran. Automation Science and Engineering, 7(3): 681--685, 2010.
- Winprop: Indoor Propagation, AWE Communications GmbH  
<http://www.awe-communications.de/Brochures/BrochureIndoor.pdf>  
<http://www.awe-communications.com/Propagation/Indoor/Empirical/index.htm>
- Wikipedia: [http://en.wikipedia.org/wiki/Wi-Fi\\_positioning\\_system](http://en.wikipedia.org/wiki/Wi-Fi_positioning_system)
- [xx] M. Worboys, M. Duckham “*GIS: a computing perspective*” (2004), CRC Press, Florida, USA
- [xx] D. Eppstein Dijkstra Recipe, retrievable: <http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/117228>.
- [xx] J. Lerner, D. Wagner, K. Zweig “*Algorithms of Large and Complex networks*” (2009), Springer, Berlin, Germany
- [xx] A. Patel “*Introduction to A\**” (2014) retrievable: <http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html>
- J.Mulligan, “A Performance Analysis of a CSMA Multihop Packet Radio Network”, Chapter 7: “Indoor Radio Propagation”, Faculty of Virginia Polytechnic Institute and State University, 1997.  
<http://scholar.lib.vt.edu/theses/available/etd-51997-22830/unrestricted/Ch7.pdf>
- Winprop: Indoor Propagation, AWE Communications GmbH  
<http://www.awe-communications.de/Brochures/BrochureIndoor.pdf>  
<http://www.awe-communications.com/Propagation/Indoor/Empirical/index.htm>
- J.Cole[blog]: <http://jasmcole.com/2014/08/25/helmhurts/>
- Wikipedia: [http://en.wikipedia.org/wiki/Wi-Fi\\_positioning\\_system](http://en.wikipedia.org/wiki/Wi-Fi_positioning_system)
- Michael Quan, Eduardo Navarro, and Benjamin Peuker, “*Wi-Fi Localization Using RSSI Fingerprinting*,” California Polytechnic State University 1 Grand Avenue, San Luis Obispo, CA, 2010.

- Vahideh Moghtadaiee, Andrew G. Dempster, “*Design protocol and performance analysis of indoor fingerprinting positioning systems*”, School of Electrical Engineering and Telecommunications, University of New South Wales, Sydney, Australia, Physical Communication, 2014.
- William Ching, Rue Jing The, Binghao Li, Chris Rizos, “*Uniwide WiFi Based Positioning System*”, Technology and Society (ISTAS), 2010 IEEE International Symposium, 2010.
- K. Benkič, M. Malajner, P. Planinšič, Ž. Čučej, “*Using RSSI value for distance estimation in Wireless sensor networks based on ZigBee*”, SPaRC Laboratory UM-FERI Maribor, 2008.
- Eladio Martin, Oriol Vinyals, Gerald Friedland, Ruzena Bajcsy, “*Precise Indoor Localization Using Smart Phones*”, International Computer Science Institute & University of California, 2010.
- Anindya S. Paul and Eric A. Wan, “*Wi-Fi Based Indoor Localization and Tracking Using Sigma-Point Kalman Filtering Methods*”, Position, Location and Navigation Symposium, IEEE/ION, 2008.
- A.T. Parameswaran, M.I.Husain, S. Upadhyaya, “*Is RSSI a Reliable Parameter in Sensor Localization Algorithms – An Experimental Study*”. 28th International Symposium On Reliable Distributed Systems, New York. September 2009.
- Holland & Holland Enterprises Ltd, n.d., <http://www.successful-project-management.com/project-risk-management.html>)
- Wikipedia: [http://en.wikipedia.org/wiki/Log-distance\\_path\\_loss\\_model](http://en.wikipedia.org/wiki/Log-distance_path_loss_model)

## Appendix

### APPENDIX 1. ANALYSIS OF THE TESTS IN THE FACULTY OF ARCHITECTURE

The set-up of the test was as follows: one scanner was placed in an open space, while two team members walked with two smartphones, in two different directions away from the scanner to the fixed points (see the blue and purple paths). At every point stayed the team members there for 5 minutes, while placing the phone on the table.



*Fig. 116. Set-up Test 1*

For test two four phones were used and the same route was traversed three times. After the first cycle was noticed that the two other Android smartphones were detected less, so for the second and the third cycle only the two smartphones of the same brand and type were used, that were also used for test 1 and 2.

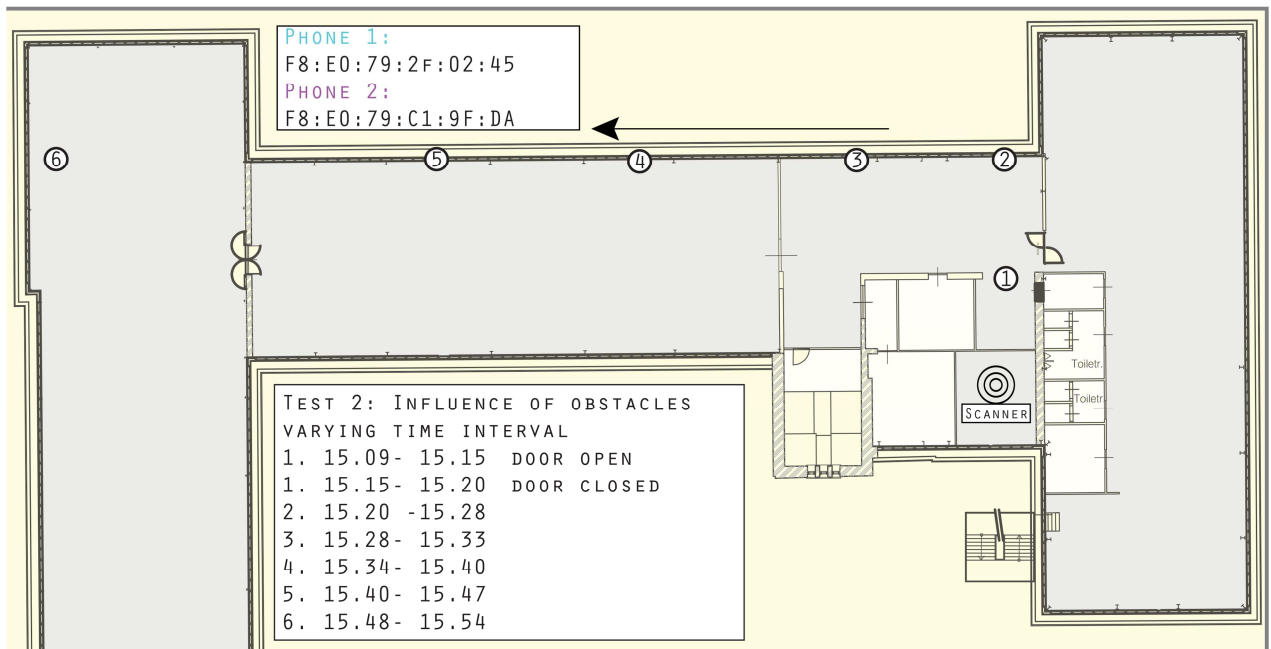


Fig. 117. Set-up test

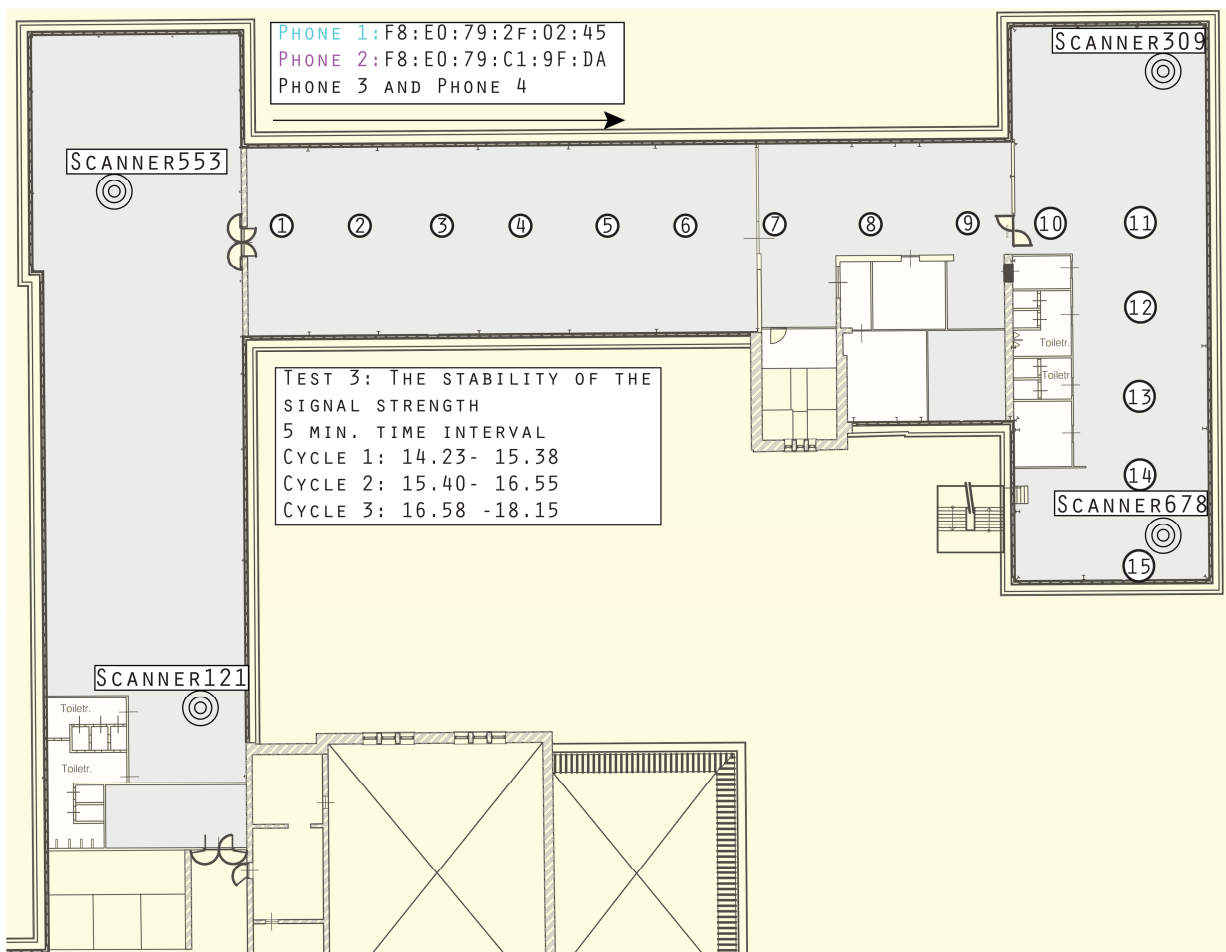
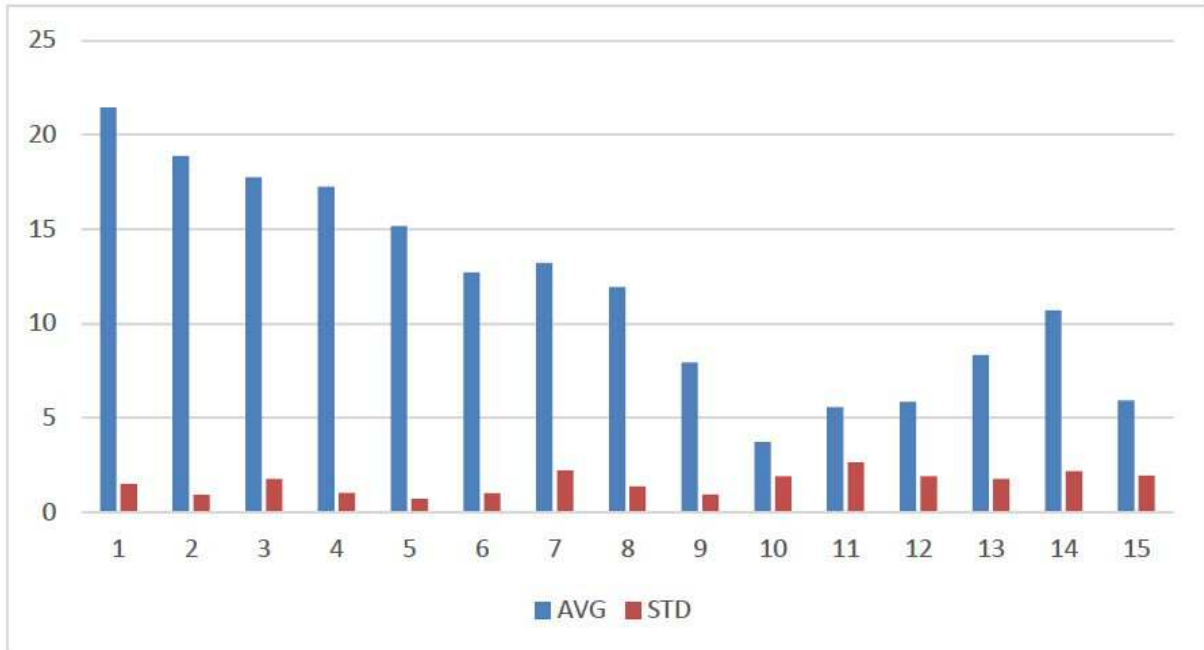


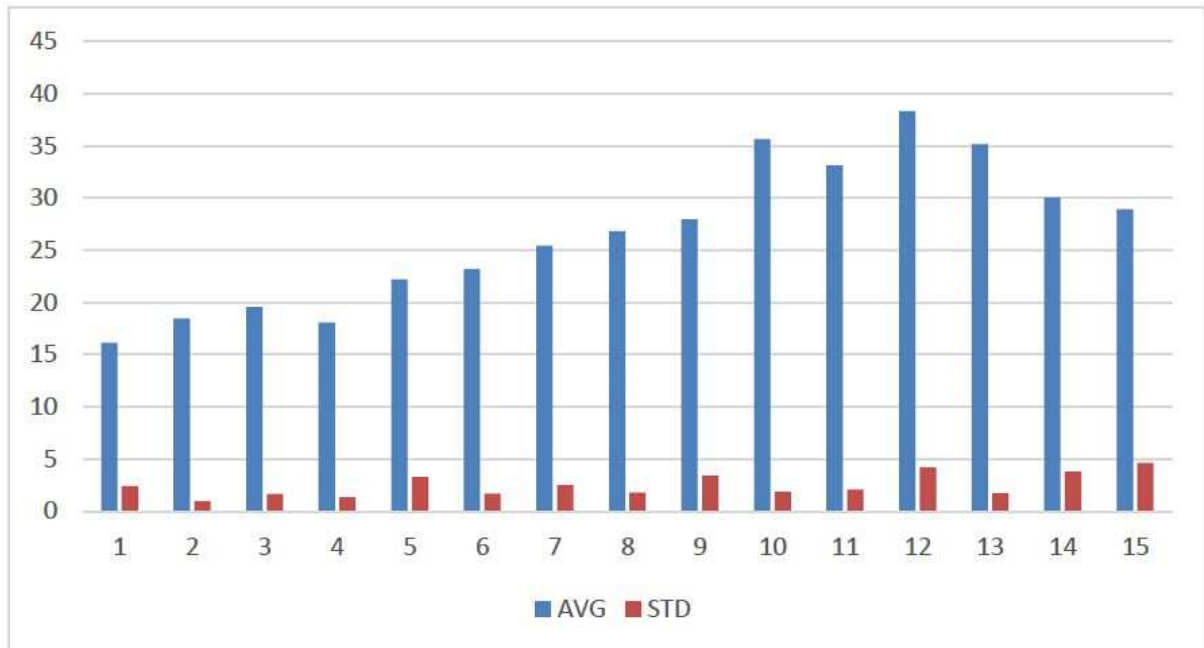
Fig. 118. Set-up test 3

Test 3, all cycles- total AVG RSSI

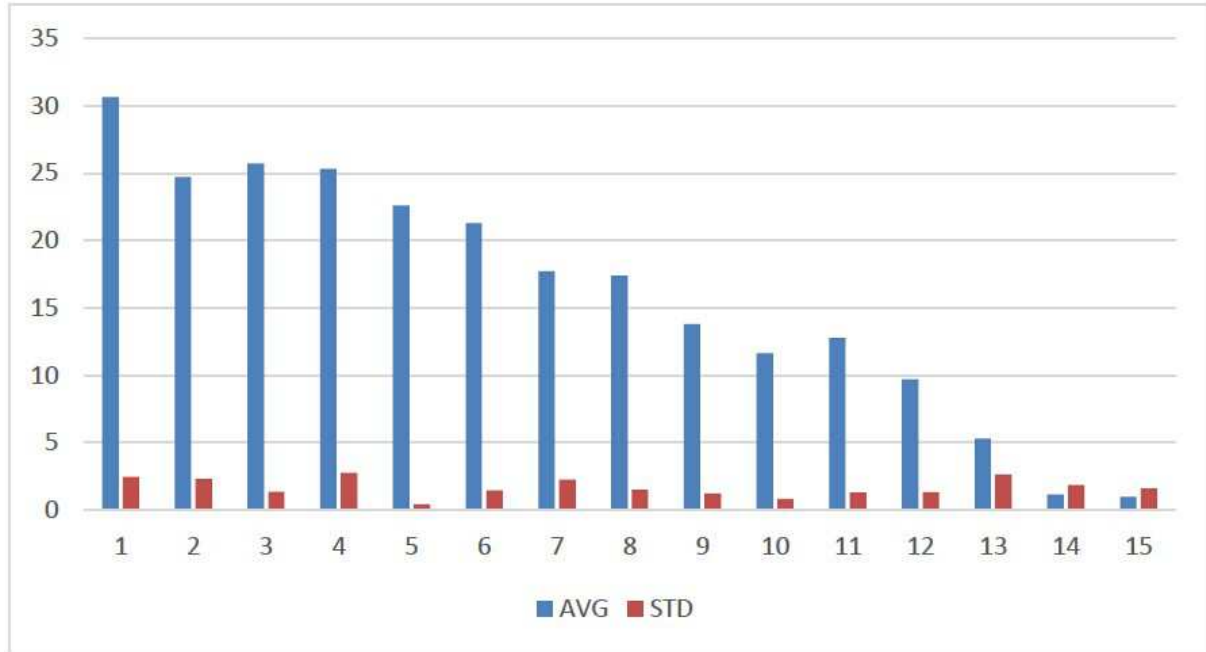
Mesh 121:



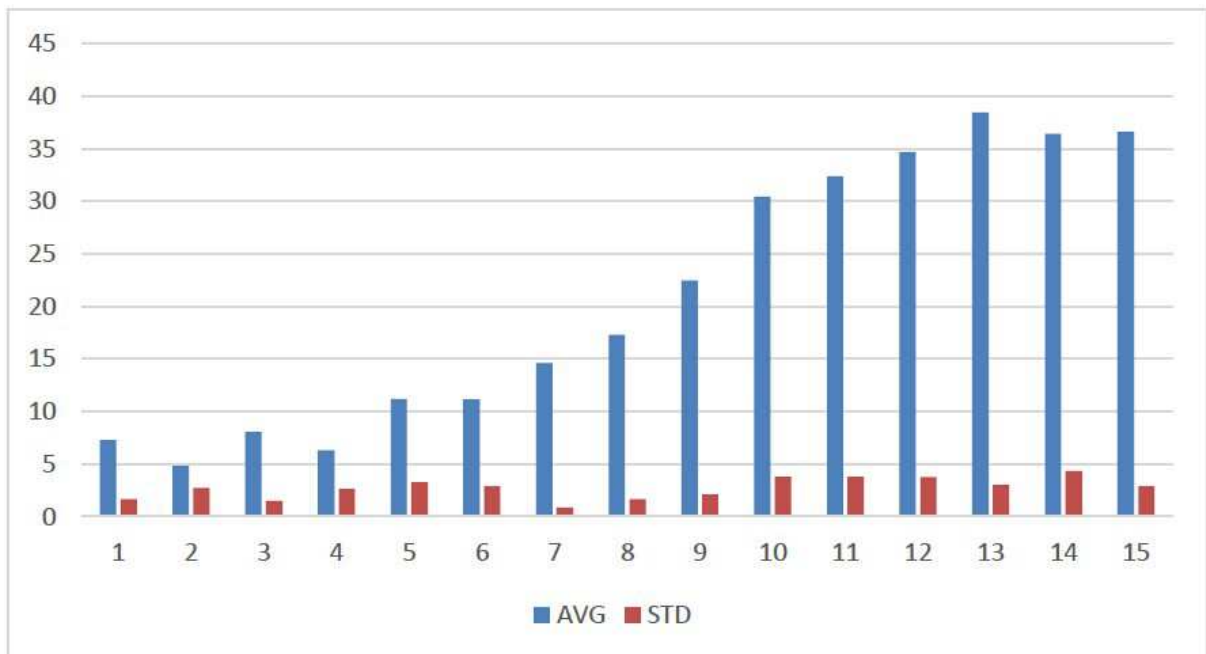
Mesh309:



Mesh 553:



Mesh678:





APPENDIX 2. TESTS 'DE ROTTERDAM' RESULTS

**Appendix I**

Average RSSI values and their counts of layout 1's first test

No.	AVG	COUNT
1	35.8667	30
2	24.5556	27
3	27.3981	26
4	27.6778	29
5	18.6667	27
6	3.5333	15
7	6	2
8	4.7976	24
9	11.7074	26
10	6.9716	24
11	4.3333	24
12	0	0
13	0	0
14	0	0
15	0	0
16	0	0
17	33	30
18	34.5556	27
19	15.3386	25
20	4.619	22
21	0.3333	2
22	0	1

No.	AVG	COUNT
1	10.6111	9
2	24.3296	29
3	22.9101	25
4	15.8571	16
5	0	0
6	0	0
7	19.6667	7
8	22.037	26
9	33.4667	30
10	33.2222	27
11	36.2222	27
12	30.8	30
13	25.3333	24
14	2	1
15	20.3333	24
16	21.3095	27
17	24.8565	26
18	17.9348	476
19	0	0
20	29.1667	5
21	32.1481	26
22	32	27

No.	AVG	COUNT
1	0	0
2	0	0
3	0	0
4	6	2
5	5.0298	22
6	9.9714	27
7	15.3796	26
8	8.7926	25
9	10.6296	27
10	5.3241	19
11	19.1074	29
12	19.7361	26
13	22.8	30
14	19.6944	26
15	30.2	30
16	33.8889	27
17	0.5556	3
18	0.3333	1
19	7.2269	26
20	12.4028	26
21	20.9954	26
22	18.4333	30

No.	AVG	COUNT
1	26.7048	26
2	23.3333	3
3	26.7667	28
4	30	27
5	34.963	27
6	27.0148	26
7	18.3667	15
8	2.3333	1
9	0	0
10	0	0
11	0	0
12	0	0
13	18.8889	7
14	19.33	24
15	11.6667	2
16	0	0
17	23.7407	29
18	21.6435	23
19	34.4	30
20	27.8333	24
21	0	0
22	0	0

\* Yellow rows indicate locations where the three phones were scanned less than ten times in total in the five minutes

(a) Mesh121

(b) Mesh309

(c) Mesh553

(d) Mesh678

## Appendix I

Average RSSI values and their counts of layout 1's second test

No.	AVG	COUNT
1	37.4074	27
2	22.1667	30
3	27.4815	27
4	22.7111	29
5	15.7407	26
6	3.0893	23
7	3	2
8	2.4444	10
9	12.8426	26
10	9.0985	24
11	3.1429	17
12	0	0
13	0	0
14	0	0
15	0	0
16	22.6667	6
17	31.1231	22
18	33.6667	27
19	16.9	30
20	3.5635	24
21	0	0
22	0	0

(a) Mesh121

No.	AVG	COUNT
1	19.0167	10
2	28.2333	30
3	27.5873	25
4	11.6667	2
5	0	0
6	13	2
7	18.0926	17
8	17.9306	21
9	29.3704	25
10	32.0667	30
11	35.3704	27
12	25.6667	27
13	24.1549	31
14	22.6667	3
15	20.463	24
16	19.2593	23
17	23.506	17
18	18.3995	22
19	0	0
20	0	0
21	30.9593	23
22	31.5648	26

(b) Mesh309

No.	AVG	COUNT
1	3	1
2	1.1667	2
3	0	0
4	7.5	4
5	8.2222	22
6	12.2667	30
7	16.7989	25
8	6.8333	15
9	5.7111	15
10	2	5
11	18.4444	27
12	22.7667	30
13	22.4333	32
14	20.5159	20
15	33.1667	30
16	30.5556	27
17	0	0
18	3	3
19	10.2917	26
20	10.8333	28
21	18.6444	23
22	18.1852	27

(c) Mesh553

No.	AVG	COUNT
1	25.0741	27
2	25.6667	3
3	23.662	26
4	30.1	30
5	33.4815	27
6	27.1889	29
7	19.2286	19
8	0	0
9	0	0
10	0	0
11	0	0
12	0	0
13	15.8611	9
14	19.9762	19
15	0	0
16	26.1667	4
17	22.2381	17
18	21.5417	22
19	34.4	30
20	25.6667	27
21	0	0
22	0	0

(d) Mesh678

\* Yellow rows indicate locations where the three phones were scanned less than ten times in total in the five minutes

## Appendix I

Average RSSI values and their counts of layout 2's first test

No.	AVG	COUNT
1	37.962963	27
2	40.9	30
3	39.037037	27
4	35.7	30
5	33.953704	26
6	19.875	23
7	16.155556	29
8	17.24537	25
9	25.333333	30
10	29.398148	26
11	27.37037	27
12	10.783333	28
13	6.3657407	25
14	2.662037	26
15	2.5666667	13
16	3.2916667	14
17	43.666667	30
18	39.074074	27
19	26.366667	30
20	14.592593	27
21	17.595238	84
22	14.9	30

No.	AVG	COUNT
1	18.64881	23
2	33.266667	30
3	29.259259	27
4	11.808333	13
5	4	2
6	0	0
7	19.285714	21
8	21.511905	23
9	32.333333	29
10	42	27
11	45.259259	27
12	29.966667	28
13	28.693122	25
14	22	4
15	21.802249	24
16	23.986111	26
17	23.941667	28
18	20.825397	25
19	0	0
20	29.5	4
21	31.833333	30
22	34.688889	29

No.	AVG	COUNT
1	0	0
2	13	6
3	6	1
4	19.333333	2
5	24.202778	27
6	30.259259	27
7	31.194444	26
8	21.736111	22
9	17.111111	13
10	21.126984	25
11	31.111111	27
12	35.703704	27
13	43.033333	30
14	35.259259	27
15	36	30
16	33.740741	27
17	0	0
18	0	0
19	19.285714	19
20	24.481481	27
21	26.159188	76
22	29.833333	26

No.	AVG	COUNT
1	29.115741	26
2	19.444444	27
3	30.244444	29
4	32.703704	27
5	37.266667	30
6	32.814815	27
7	28.518519	27
8	17.314815	28
9	12.226852	25
10	4.4074074	25
11	2.2888889	23
12	6.662037	24
13	20.433333	29
14	18.777778	27
15	13.866667	29
16	13.203704	24
17	27.819444	26
18	29.2	30
19	38.074074	27
20	33.666667	27
21	26.144349	81
22	5.212963	25

\* Yellow rows indicate locations where the three phones were scanned less than ten times in total in the five minutes

(a) Mesh121

(b) Mesh309

(c) Mesh553

(d) Mesh678

## Appendix I

Average RSSI values and their counts of layout 2's second test

No.	AVG	COUNT
1	35.592593	27
2	37.62963	27
3	40.966667	30
4	35.333333	27
5	32.8	30
6	16.888889	27
7	13.875	26
8	12.533333	30
9	34.148148	27
10	24.933333	30
11	23.851852	27
12	11.777778	27
13	3.8490741	27
14	3.2	19
15	4.5846561	22
16	4.8118687	28
17	38.37037	27
18	41.466667	29
19	26.222222	27
20	16.282407	26
21	13.114815	29
22	13.62963	27

(a) Mesh121

No.	AVG	COUNT
1	22.655556	16
2	33.9	30
3	24.629108	426
4	24.208333	140
5	0	0
6	12.333333	3
7	18.75	20
8	21.222222	25
9	34.281481	28
10	46.925926	27
11	46.592593	27
12	36.533333	30
13	29.189815	26
14	21.025	27
15	27.152778	26
16	29.236364	31
17	25.511111	29
18	22.62963	26
19	20.333333	3
20	0	0
21	34.800926	25
22	34.592593	27

(b) Mesh309

No.	AVG	COUNT
1	1	1
2	9.8333333	3
3	2	1
4	7.3333333	1
5	24.740741	27
6	32.733333	30
7	32.460317	25
8	22.416667	22
9	18.761905	20
10	26.648148	23
11	36.555556	27
12	41.466667	29
13	55	27
14	39.9	30
15	41.518519	27
16	38.515152	33
17	3	1
18	0	0
19	21.422619	23
20	22.435185	24
21	31.522222	29
22	30.069444	26

(c) Mesh553

No.	AVG	COUNT
1	30	30
2	18.097222	26
3	29.851852	27
4	33.466667	30
5	44.592593	27
6	36.9	30
7	25.703704	27
8	16.648148	26
9	11.903704	28
10	3.5119048	22
11	3.6686508	24
12	7.125	26
13	22.097222	26
14	19.840741	29
15	16.884259	26
16	12.787879	33
17	28.066667	30
18	26.018519	26
19	35.666667	30
20	36.444444	27
21	10.050926	25
22	4.9603175	23

(d) Mesh678

\* Yellow rows indicate locations where the three phones were scanned less than ten times in total in the five minutes

## Appendix II

Average RSSI values and their counts of layout 2's second test

Test 1	Test 2	AVG
35.8667	37.4074	36.637
24.5556	22.1667	23.3611
27.3981	27.4815	27.4398
27.6778	22.7111	25.1944
18.6667	15.7407	17.2037
3.5333	3.0893	3.3113
6	3	0
4.7976	2.4444	3.621
11.7074	12.8426	12.275
6.9716	9.0985	8.0351
4.3333	3.1429	3.7381
0	0	0
0	0	0
0	0	0
0	0	0
0	22.6667	0
33	31.1231	32.0616
34.5556	33.6667	34.1111
15.3386	16.9	16.1193
4.619	3.5635	4.0913
0.3333	0	0
0	0	0

(a) Mesh121

Test 1	Test 2	AVG
10.6111	19.0167	0
24.3296	28.2333	26.2815
22.9101	27.5873	25.2487
15.8571	11.6667	0
0	0	0
0	13	0
19.6667	18.0926	0
22.037	17.9306	19.9838
33.4667	29.3704	31.4185
33.2222	32.0667	32.6444
36.2222	35.3704	35.7963
30.8	25.6667	28.2333
25.3333	24.1549	24.7441
2	22.6667	0
20.3333	20.463	20.3981
21.3095	19.2593	20.2844
24.8565	23.506	24.1812
17.9348	18.3995	18.1671
0	0	0
29.1667	0	0
32.1481	30.9593	31.5537
32	31.5648	31.7824

(b) Mesh309

Test 1	Test 2	AVG
0	3	0
0	1.1667	0
0	0	0
6	7.5	0
5.0298	8.2222	6.626
9.9714	12.2667	11.119
15.3796	16.7989	16.0893
8.7926	6.8333	7.813
10.6296	5.7111	8.1704
5.3241	2	0
19.1074	18.4444	18.7759
19.7361	22.7667	21.2514
22.8	22.4333	22.6167
19.6944	20.5159	20.1052
30.2	33.1667	31.6833
33.8889	30.5556	32.2222
0.5556	0	0
0.3333	3	0
7.2269	10.2917	8.7593
12.4028	10.8333	11.6181
20.9954	18.6444	19.8199
18.4333	18.1852	18.3093

(c) Mesh553

Test 1	Test 2	AVG
26.7048	25.0741	25.8894
23.3333	25.6667	0
26.7667	23.662	25.2144
30	30.1	30.05
34.963	33.4815	34.2222
27.0148	27.1889	27.1019
18.3667	19.2286	18.7976
2.3333	0	0
0	0	0
0	0	0
0	0	0
0	0	0
18.8889	15.8611	0
19.33	19.9762	19.6531
11.6667	0	0
0	26.1667	0
23.7407	22.2381	22.9894
21.6435	21.5417	21.5926
34.4	34.4	34.4
27.8333	25.6667	26.75
0	0	0
0	0	0

(d) Mesh678

\* Yellow cells indicate locations where the three phones were scanned less than ten times in total in the five minutes

Appendix III

Final input data of layout1 in 4 by 4 grid

No.	x	y	AVG
1	2	2	36.637
2	4	4	23.3611
3	2	4	27.4398
4	2	6	25.1944
5	2	8	17.2037
6	4	8	3.3113
7	5	8	0
8	5	6	3.621
9	5	4	12.275
10	7	4	8.0351
11	9	4	3.7381
12	9	6	0
13	9	8	0
14	7	8	0
15	9	10	0
16	9	12	0
17	1	5	32.0616
18	2	3	34.1111
19	3	9	16.1193
20	4	9	4.0913
21	10	6	0
22	10	5	0

(a) Mesh121

No.	x	y	AVG
1	2	2	0
2	4	4	26.2815
3	2	4	25.2487
4	2	6	0
5	2	8	0
6	4	8	0
7	5	8	0
8	5	6	19.9838
9	5	4	31.4185
10	7	4	32.6444
11	9	4	35.7963
12	9	6	28.2333
13	9	8	24.7441
14	7	8	0
15	9	10	20.3981
16	9	12	20.2844
17	1	5	24.1812
18	2	3	18.1672
19	3	9	0
20	4	9	0
21	10	6	31.5537
22	10	5	31.7824

(b) Mesh309

No.	x	y	AVG
1	2	2	0
2	4	4	0
3	2	4	0
4	2	6	0
5	2	8	6.626
6	4	8	11.119
7	5	8	16.0893
8	5	6	7.813
9	5	4	8.1704
10	7	4	0
11	9	4	18.7759
12	9	6	21.2514
13	9	8	22.6167
14	7	8	20.1052
15	9	10	31.6833
16	9	12	32.2222
17	1	5	0
18	2	3	0
19	3	9	8.7593
20	4	9	11.6181
21	10	6	19.8199
22	10	5	18.3093

(c) Mesh553

No.	x	y	AVG
1	2	2	25.8894
2	4	4	0
3	2	4	25.2144
4	2	6	30.05
5	2	8	34.2222
6	4	8	27.1019
7	5	8	18.7976
8	5	6	0
9	5	4	0
10	7	4	0
11	9	4	0
12	9	6	0
13	9	8	0
14	7	8	19.6531
15	9	10	0
16	9	12	0
17	1	5	22.9894
18	2	3	21.5926
19	3	9	34.4
20	4	9	26.75
21	10	6	0
22	10	5	0

(d) Mesh678

### Appendix III

Final input data of layout1 in 2 by 2 grid

No.	x	y	AVG
1	4	4	36.637
2	7	8	23.3611
3	4	8	27.4398
4	4	11	25.1944
5	4	15	17.2037
6	7	15	3.3113
7	10	15	0
8	10	11	3.621
9	10	8	12.275
10	14	8	8.0351
11	18	8	3.7381
12	18	11	0
13	18	15	0
14	14	15	0
15	18	19	0
16	18	23	0
17	2	9	32.0616
18	3	5	34.1111
19	6	17	16.1193
20	8	17	4.0913
21	20	12	0
22	20	10	0

(a) Mesh121

No.	x	y	AVG
1	4	4	0
2	7	8	26.2815
3	4	8	25.2487
4	4	11	0
5	4	15	0
6	7	15	0
7	10	15	0
8	10	11	19.9838
9	10	8	31.4185
10	14	8	32.6444
11	18	8	35.7963
12	18	11	28.2333
13	18	15	24.7441
14	14	15	0
15	18	19	20.3981
16	18	23	20.2844
17	2	9	24.1812
18	3	5	18.1672
19	6	17	0
20	8	17	0
21	20	12	31.5537
22	20	10	31.7824

(b) Mesh309

No.	x	y	AVG
1	4	4	0
2	7	8	0
3	4	8	0
4	4	11	0
5	4	15	6.626
6	7	15	11.119
7	10	15	16.0893
8	10	11	7.813
9	10	8	8.1704
10	14	8	0
11	18	8	18.7759
12	18	11	21.2514
13	18	15	22.6167
14	14	15	20.1052
15	18	19	31.6833
16	18	23	32.2222
17	2	9	0
18	3	5	0
19	6	17	8.7593
20	8	17	11.6181
21	20	12	19.8199
22	20	10	18.3093

(c) Mesh553

No.	x	y	AVG
1	4	4	25.8894
2	7	8	0
3	4	8	25.2144
4	4	11	30.05
5	4	15	34.2222
6	7	15	27.1019
7	10	15	18.7976
8	10	11	0
9	10	8	0
10	14	8	0
11	18	8	0
12	18	11	0
13	18	15	0
14	14	15	19.6531
15	18	19	0
16	18	23	0
17	2	9	22.9894
18	3	5	21.5926
19	6	17	34.4
20	8	17	26.75
21	20	12	0
22	20	10	0

(d) Mesh678

### Appendix III

Final input data of layout2 in 4 by 4 grid

No.	x	y	AVG
1	2	2	36.7778
2	4	4	39.2648
3	2	4	40.0019
4	2	6	35.5167
5	2	8	33.3769
6	4	8	18.3819
7	5	8	15.0153
8	5	6	14.8894
9	5	4	29.7407
10	7	4	27.1657
11	9	4	25.6111
12	9	6	11.2806
13	9	8	5.1074
14	7	8	2.931
15	9	10	3.5757
16	9	12	4.0518
17	1	5	41.0185
18	2	3	40.2704
19	3	9	26.2944
20	4	9	15.4375
21	10	6	15.355
22	10	5	14.2648

(a) Mesh121

No.	x	y	AVG
1	2	2	20.6522
2	4	4	33.5833
3	2	4	26.9442
4	2	6	18.0083
5	2	8	0
6	4	8	0
7	5	8	19.0179
8	5	6	21.3671
9	5	4	33.3074
10	7	4	44.463
11	9	4	45.9259
12	9	6	33.25
13	9	8	28.9415
14	7	8	0
15	9	10	24.4775
16	9	12	26.6112
17	1	5	24.7264
18	2	3	21.7275
19	3	9	0
20	4	9	0
21	10	6	33.3171
22	10	5	34.6407

(b) Mesh309

No.	x	y	AVG
1	2	2	0
2	4	4	0
3	2	4	0
4	2	6	0
5	2	8	24.4718
6	4	8	31.4963
7	5	8	31.8274
8	5	6	22.0764
9	5	4	17.9365
10	7	4	23.8876
11	9	4	33.8333
12	9	6	38.5852
13	9	8	49.0167
14	7	8	37.5796
15	9	10	38.7593
16	9	12	36.1279
17	1	5	0
18	2	3	0
19	3	9	20.3542
20	4	9	23.4583
21	10	6	28.8407
22	10	5	29.9514

(c) Mesh553

No.	x	y	AVG
1	2	2	29.5579
2	4	4	18.7708
3	2	4	30.0481
4	2	6	33.0852
5	2	8	40.9296
6	4	8	34.8574
7	5	8	27.1111
8	5	6	16.9815
9	5	4	12.0653
10	7	4	3.9597
11	9	4	2.9788
12	9	6	6.8935
13	9	8	21.2653
14	7	8	19.3093
15	9	10	15.3755
16	9	12	12.9958
17	1	5	27.9431
18	2	3	27.6093
19	3	9	36.8704
20	4	9	35.0556
21	10	6	18.0976
22	10	5	5.0866

(d) Mesh678



### Appendix III

Final input data of layout2 in 2 by 2 grid

No.	x	y	AVG
1	4	4	36.7778
2	7	8	39.2648
3	4	8	40.0019
4	4	11	35.5167
5	4	15	33.3769
6	7	15	18.3819
7	10	15	15.0153
8	10	11	14.8894
9	10	8	29.7407
10	14	8	27.1657
11	18	8	25.6111
12	18	11	11.2806
13	18	15	5.1074
14	14	15	2.931
15	18	19	3.5757
16	18	23	4.0518
17	2	9	41.0185
18	3	5	40.2704
19	6	17	26.2944
20	8	17	15.4375
21	20	12	15.355
22	20	10	14.2648

(a) Mesh121

No.	x	y	AVG
1	4	4	20.6522
2	7	8	33.5833
3	4	8	26.9442
4	4	11	18.0083
5	4	15	0
6	7	15	0
7	10	15	19.0179
8	10	11	21.3671
9	10	8	33.3074
10	14	8	44.463
11	18	8	45.9259
12	18	11	33.25
13	18	15	28.9415
14	14	15	0
15	18	19	24.4775
16	18	23	26.6112
17	2	9	24.7264
18	3	5	21.7275
19	6	17	0
20	8	17	0
21	20	12	33.3171
22	20	10	34.6407

(b) Mesh309

No.	x	y	AVG
1	4	4	0
2	7	8	0
3	4	8	0
4	4	11	0
5	4	15	24.4718
6	7	15	31.4963
7	10	15	31.8274
8	10	11	22.0764
9	10	8	17.9365
10	14	8	23.8876
11	18	8	33.8333
12	18	11	38.5852
13	18	15	49.0167
14	14	15	37.5796
15	18	19	38.7593
16	18	23	36.1279
17	2	9	0
18	3	5	0
19	6	17	20.3542
20	8	17	23.4583
21	20	12	28.8407
22	20	10	29.9514

(c) Mesh553

No.	x	y	AVG
1	4	4	29.5579
2	7	8	18.7708
3	4	8	30.0481
4	4	11	33.0852
5	4	15	40.9296
6	7	15	34.8574
7	10	15	27.1111
8	10	11	16.9815
9	10	8	12.0653
10	14	8	3.9597
11	18	8	2.9788
12	18	11	6.8935
13	18	15	21.2653
14	14	15	19.3093
15	18	19	15.3755
16	18	23	12.9958
17	2	9	27.9431
18	3	5	27.6093
19	6	17	36.8704
20	8	17	35.0556
21	20	12	18.0976
22	20	10	5.0866

(d) Mesh678

## APPENDIX 3. SPACE SUBDIVISION

Python 2.7 code for the Semi- automatic space subdivision

```
import shapefile
import shapely
```

```
def subdivision():
```

```
    from shapely.geometry import Point,LineString,Polygon
```

```
    fh = open ("newtriangulation2.1.node")
```

```
    contentsnode = fh.readlines()[1:]
```

```
    fh.close()
```

```
    index_list = []
```

```
    for line in contentsnode:
```

```
        if line.startswith("#"):
```

```
            continue
```

```
        else:
```

```
            linesplit = line[8:].split(" ")
```

```
            x = (linesplit[0])
```

```
            y = (linesplit[1])
```

```
            coordinates = (x,y)
```

```
            index_list.append(coordinates)
```

```
    trianglelist = []
```

```
    fh = open ("newtriangulation2.1.ele")
```

```
    contentsele = fh.readlines()[1:]
```

```
    fh.close()
```

```
    for line in contentsele:
```

```
        if line.startswith("#"):
```

```
            continue
```

```
        else:
```

```
            linestrip = line.strip()
```

```
            linesplit = linestrip.split(" ")
```

```
            line = []
```

```
            for item in linesplit:
```

```
                if item == ":
```

```
                    continue
```

```
                else: line.append(item)
```

```
            index1 = int(line[1]) -1
```

```
            index2 = int(line[2]) -1
```

```
            index3 = int(line[3]) -1
```

```
            pt1 = (index_list[index1])
```

```
            f1= float(pt1[0])
```

```
            f2 = float(pt1[1])
```

```
            pt2 = (index_list[index2])
```

```
            f3 = float(pt2[0])
```

```
            f4 = float(pt2[1])
```

```

pt3 = (index_list[index3])
f5 = float(pt3[0])
f6 = float(pt3[1])

npt1 = (f1,f2)
npt2 = (f3,f4)
npt3 = (f5,f6)
points = [npt1,npt2, npt3]
triangle = Polygon(points)
trianglelist.append(triangle)
l1 = [npt1,npt2]
l2 = [npt2,npt3]
l3 = [npt3,npt1]
line1 = LineString(l1)
line2 = LineString(l2)
line3 = LineString(l3)

"""point of the scanners for layout 1:"""
scanner121 = Point(2.736317, 2.296404)
scanner678 = Point(2.736317, 33.771274)
scanner553 = Point(40.260276, 43.821449)
scanner309 = Point(40.260276, 9.820339)

"""check wich triangles are in a range of 10m of the four scanners"""
list121 = []
q = 0
while q <len(trianglelist):
    if trianglelist[q].intersects(scanner121.buffer(10)):
        list121.append(trianglelist[q])
        q = q+1
    else:
        q = q+1

list678 = []
r = 0
while r <len(trianglelist):
    if trianglelist[r].intersects(scanner678.buffer(10)):
        list678.append(trianglelist[r])
        r = r+1
    else:
        r = r+1

""" remove double triangles"""
g = 0
while g <len(list678):
    h = 0
    while h < len(list121):
        if list121[h]==list678[g]:
            list121.remove(list678[g])

```

```

        h = h+1
    else:
        h = h+1
    g = g+1

# merge triangles
from shapely.ops import cascaded_union
sub121 = cascaded_union(list121)

# merge triangles
from shapely.ops import cascaded_union
sub678 = cascaded_union(list678)

list553= []
s = 0
while s <len(trianglelist):
    if trianglelist[s].intersects(scanner553.buffer(10)):
        list553.append(trianglelist[s])
        s = s+1
    else:
        s = s+1
# merge triangles
from shapely.ops import cascaded_union
sub553 = cascaded_union(list553)

list309= []
t = 0
while t <len(trianglelist):

    if trianglelist[t].intersects(scanner309.buffer(10)):
        list309.append(trianglelist[t])
        t = t+1
    else:
        t = t+1
# merge triangles
from shapely.ops import cascaded_union
sub309 = cascaded_union(list309)

""remove triangles that fall in range of the scanners from the trianglelist""
u = 0
while u <len(list121):
    v = 0
    while v < len(trianglelist):
        if trianglelist[v]==list121[u]:
            trianglelist.remove(trianglelist[v])
            v = v+1
        else:
            v = v+1

```

```

    u = u+1

w = 0
while w <len(list678):
    x = 0
    while x < len(trianglelist):
        if trianglelist[x]==list678[w]:
            trianglelist.remove(trianglelist[x])
            x = x+1
        else:
            x = x+1
    w = w+1

y = 0
while y <len(list553):
    z = 0
    while z < len(trianglelist):
        if trianglelist[z]==list553[y]:
            trianglelist.remove(trianglelist[z])
            z = z+1
        else:
            z = z+1
    y = y+1

a = 0
while a <len(list309):
    b = 0
    while b < len(trianglelist):
        if trianglelist[b]==list309[a]:
            trianglelist.remove(trianglelist[b])
            b = b+1
        else:
            b = b+1
    a = a+1

"""make subdivisions between the scanner subdivisions"""
buffer121 = []
c = 0
while c <len(trianglelist):
    if trianglelist[c].intersects(scanner121.buffer(20)):
        buffer121.append(trianglelist[c])
        c = c+1
    else:
        c = c+1

buffer309 = []
d = 0
while d <len(trianglelist):
    if trianglelist[d].intersects(scanner309.buffer(19)):

```

```

    buffer309.append(trianglelist[d])
    d = d+1
else:
    d = d+1

buffer678 = []
e = 0
while e <len(trianglelist):
    if trianglelist[e].intersects(scanner678.buffer(19)):
        buffer678.append(trianglelist[e])
        e = e+1
    else:
        e = e+1

buffer553 = []
f = 0
while f <len(trianglelist):
    if trianglelist[f].intersects(scanner553.buffer(20)):
        buffer553.append(trianglelist[f])
        f = f+1
    else:
        f = f+1

"""Remove double triangles from the subdivisions, so they all be around the same size"""
g = 0
while g <len(buffer309):
    h = 0
    while h < len(buffer553):
        if buffer309[g]==buffer553[h]:
            buffer553.remove(buffer309[g])
            h = h+1
        else:
            h = h+1
    g = g+1

g = 0
while g <len(buffer309):
    h = 0
    while h < len(buffer121):
        if buffer309[g]==buffer121[h]:
            buffer309.remove(buffer121[h])
            h = h+1
        else:
            h = h+1
    g = g+1

g = 0
while g <len(buffer678):
    h = 0

```

```

while h < len(buffer121):
    if buffer678[g]==buffer121[h]:
        buffer678.remove(buffer121[h])
        h = h+1
    else:
        h = h+1
    g = g+1

g = 0
while g <len(buffer553):
    h = 0
    while h < len(buffer678):
        if buffer553[g]==buffer678[h]:
            buffer553.remove(buffer678[h])
            h = h+1
        else:
            h = h+1
    g = g+1

g = 0
while g <len(buffer678):
    h = 0
    while h < len(buffer309):
        if buffer678[g]==buffer309[h]:
            buffer309.remove(buffer678[g])
            h = h+1
        else:
            h = h+1
    g = g+1

"""make one polygon of the triangles"""
from shapely.ops import cascaded_union
sub309_553 = cascaded_union(buffer309)

from shapely.ops import cascaded_union
sub678_553 = cascaded_union(buffer553)

from shapely.ops import cascaded_union
sub121_678 = cascaded_union(buffer678)

from shapely.ops import cascaded_union
sub121_309 = cascaded_union(buffer121)

```

### APPENDIX 3: NETWORK

Python 2.7 code for the Network semi- automatic (with basepoints) for the range subdivision

```
import shapefile
import shapely

def test_network(filename):
    from shapely.geometry import Point, LineString, Polygon
    sf = shapefile.Reader(filename)
    spaces = sf.shapes()
    sublist = []
    for s in spaces:
        spoly = Polygon(s.points)
        sublist.append(spoly)

    """get centroids of subdivisions"""
    clist = []
    i = 0
    while i < len(sublist):
        div = sublist[i].centroid
        clist.append(div)
        i = i+1

    """Make Point objects from the points around the core(basepoints) and make lines between
    them"""
    blist=[]
    sf = shapefile.Reader("newestbasepoints.shp")
    bpoints = sf.records()
    for b in bpoints:
        ptx = b[1]
        pty = b[2]
        newpoint = Point(ptx,pty)
        blist.append(newpoint) # list with basepoints around core

    baselinelist = []
    n = 0
    while n< len(blist):
        if n+1 < len (blist):
            baseedge = blist[n], blist[n+1]
            n = n+1
        else:
            baseedge = blist[n-1], blist[0]
            n = n+1
        line = LineString(baseedge)
        baselinelist.append(line) # list with linestrings around core

    lastedge = blist[2], blist[6], blist[5]
    lastline = LineString(lastedge)
```



```

baselinelist.append(lastline)

bpoly = [(7.557902, 14.571654), (7.603434, 29.528758),(34.808517, 29.540141),
(35.036175, 14.844843)]
basepoly = Polygon(bpoly)

"""remove centerpoints inside basepolygon"""
newclist=[]
for c in clist:
    if basepoly.buffer(0.5).contains(c):
        continue
    else:
        newclist.append(c)

"""connect centerpoints with the closest basepoint"""
minlist = []
j = 0
while j < len(newclist):
    partlist3 = []
    k = 0
    while k < len(blist):
        dis = newclist[j].distance(blist[k]), newclist[j], blist[k]
        partlist3.append(dis)
        k = k+1
    mind = min(partlist3)
    minlist.append(mind)
    j = j+1

newlinelist = []
for m in minlist:
    if m[0] > 10:
        continue
    else:
        newedge = m[1],m[2]
        newline = LineString(newedge)
        newlinelist.append(newline)

"""find neighbouring polygons"""
nlist = []
a = 0
while a < len(sublist):
    partlist = []
    partlist.append(a)
    b = 0
    while b < len(sublist):
        if not sublist[a].intersection(sublist[b]).is_empty and not a==b and not
sublist[a].intersection(sublist[b]).type == 'Point':
            partlist.append(b)
            b = b+1

```

```

        else:
            b = b+1
        nlist.append(partlist)
        a = a+1

"""make lines between neighbouring polygons"""
networklist = []
linelist = []
d = 0
while d < len(nlist):
    part2list = []
    part2list.append(d)
    e = 0
    while e < len(nlist[d]):
        edge = [clist[nlist[d][0]], clist[nlist[d][e]]]
        part2list.append(edge)
        line = LineString(edge)
        linelist.append(line)
        e = e+1
    networklist.append(part2list)
    d = d+1

"""remove the lines that are intersecting with the lines around the core"""
removelist = []
g = 0
while g < len(baselinelist):
    h = 0
    while h < len(linelist):
        if baselinelist[g].intersects(linelist[h]):
            removelist.append(linelist[h])
            h = h + 1
        else:
            h = h+1
    g = g+1

u = 0
while u < len(removelist):
    v = 0
    while v < len(linelist):
        if linelist[v]==removelist[u]:
            linelist.remove(linelist[v])
            v = v+1
        else:
            v = v+1
    u = u+1

```

## Python 2.7 code for Automatic network for the range subdivision

```
import shapefile
import shapely

def test_network(filename):
    from shapely.geometry import Point,LineString,Polygon

    sf = shapefile.Reader(filename)
    polygons = sf.records()
    spaces = sf.shapes()

    sublist = []
    for s in spaces:
        spoly = Polygon(s.points)
        sublist.append(spoly)

    """get centroids of subdivisions"""
    clist = []
    i = 0
    while i < len(sublist):
        div = i, sublist[i].centroid
        clist.append(div)
        i = i+1

    """find neighbours of the polygons and the middle of their intersecting lines,
    make the network"""
    nlist = []
    interlist = []
    linelist = []
    a = 0
    while a < len(sublist):
        partlist = []
        partlist.append(a)
        ilit = []
        ilit.append(a)
        b = 0
        while b< len(sublist):
            if not sublist[a].intersection(sublist[b]).is_empty and not a==b and not
sublist[a].intersection(sublist[b]).type == 'Point':
                partlist.append(b)
                intersection = sublist[a].intersection(sublist[b])
                midpoint = intersection.interpolate(intersection.length/2)
                ilit.append(midpoint)
                edge = clist[a][1], midpoint

                line = LineString(edge)
                print line
```

```

        linelist.append(line)# list with the lines between the center points and the midpoints
of the intersecting lines
        b = b+1
    else:
        b = b+1
    nlist.append(partlist) # list with the polygons and their neighbours
    interlist.append(ilst) # list with the midpoints of the intersecting lines
    a = a+1

```

```

"""remove point linestrings"""

```

```

doublelist = []
f = 0
while f < len(linelist):
    if linelist[f].length < 1:
        doublelist.append(linelist[f])
        f = f+1
    else:
        f = f+1

u = 0
while u < len(doublelist):
    v = 0
    while v < len(linelist):
        if linelist[v]==doublelist[u]:
            linelist.remove(linelist[v])
            v = v+1
        else:
            v = v+1
    u = u+1

```

## APPENDIX 4. PYTHON 2.7 CODE: MULTILATERATION SCRIPTS

### Multilateration.py [main script for finding a solution given scanner positions and a mac address]

```
import MySQLdb #download @ http://www.codegood.com/archives/129
from math import log, e
import sys
import shapely.geometry as g
import fiona
import intersections

#Weighted average based on time:
def wavgrssi(lstdata):
    return sum([v[0]*v[1] for v in lstdata]) / sum([v[1] for v in lstdata])

#Trustability based on how "old" the results are:
def trust(tdSEC,tinterMIN):
    if tinterMIN > 30:
        p = 1
    else:
        p = 15.0 / pow(log(tinterMIN+1,e),2)
    trustability = 1.0 - pow(log(tdSEC+1,tinterMIN*60+1),p) #+1 to avoid log(0)
    return trustability

#Localization radius based on translating RSSI values into distance
def localize(rssi, distFromCentr):
    Ro = 60.0 #RSSI at 1m distance
    R = float(rssi)
    #note: the thought was to parametrize the no value, based on the distance
    #      of the scanner to the center of the building. Outcome were values
    #      that didn't vary much from a static 'no' value.
    #no = 3.62 - log(distFromCentr,Ro)
    no = 2.8
    n = no + log(Ro/R,Ro/2) #path-loss exponent
    x = (Ro-R) / (10*n)
    d = pow(10,x)
    return d

#Error function - defines thickness of distance localization rings
def calcerr(rssi):
    #Ro = 60.0 #dBm at 1m distance
    #err = log(Ro/float(rssi),2)
    #Error could be the above - instead, consider 2.0dBm differences
    err = abs(localize(rssi,0) - localize(rssi+2.0,0))
    return err

#Not really necessary, but printable info per single run
def printformat(fin):
    fin = int(fin)
    if fin < 10:
        fout = '0{0}'.format(fin)
    else:
        fout = '{0}'.format(fin)
    return fout

#Needed only in single run - for representation purposes
def represent(building,MeshPositions,leftoverings):
    #REPRESENTATION: (.shp)
    # Define a polygon feature geometry - general schema
    polyschema = {
        'geometry': 'Polygon',
        'properties': {'id': 'int'},
    }
    pointschema = {
        'geometry': 'Point',
        'properties': {'id': 'int'},
    }
    #REPRESENT BUILDING:
    with fiona.open('derot.shp', 'w', 'ESRI Shapefile', polyschema) as c:
        c.write({
            'geometry': g.mapping(building),
            'properties': {'id': 0},
```

```

    })
    c.close()
#REPRESENT MESHLIUM SCANNER POSITIONS:
with fiona.open('MeshPositions.shp', 'w', 'ESRI Shapefile', pointschema) as c:
    for i in range(len(MeshPositions)):
        c.write({
            'geometry': g.mapping(g.Point(MeshPositions[i])),
            'properties': {'id': i},
        })
    c.close()
#REPRESENT RINGS:
with fiona.open('timeboxZ.shp', 'w', 'ESRI Shapefile', polyschema) as c:
    for i in range(len(leftoverings)):
        if leftoverings[i]:
            c.write({
                'geometry': g.mapping(leftoverings[i][0]),
                'properties': {'id': i},
            })
    c.close()
return True

# MAIN FUNCTION
def main(meshnames,MeshPositions,cellMAC,timeNOW,TIMEinterval_min,saveshp,printinfo):

    if TIMEinterval_min <= 0:
        print "Time interval cannot be 0 or less"
        sys.exit()

    # Open EXTERNAL database connection
    db = MySQLdb.connect(host="libelium.tudelft.nl",user="meshlium",
                        passwd="liuliu",db="meshliumdb" )
    # prepare a cursor object
    cursor = db.cursor()

    if printinfo:
        print "\n\n*** CELLPHONE MAC: ",cellMAC,"\nTIME NOW: ", timeNOW, "\nTIME TO SEARCH:
{0} MINUTES AGO".format(TIMEinterval_min)
    wRSSIs = [] # will contain weighted RSSI values per scanner
    for mesh in meshnames:
        RSSIs_n_TIMEtrust = []
        if printinfo:
            print "\n",mesh
#TODO:time "now" in datetime format (POSSIBLY DELETE NEXT 3 LINES LATER)
        tsql = "SELECT timestamp('{0}')" .format(timeNOW)
        cursor.execute(tsql)
        tnow = cursor.fetchone()[0]

        #SQL query to get the RSSI values and datetimes
        sql = ""SELECT RSSI, timestamp
        FROM test.{0}
        WHERE MAC='{1}' AND RSSI > 4
        AND (timestamp between '{2}' - interval {3} second and '{2}')"
        ORDER BY timestamp DESC"" .format(mesh, cellMAC, timeNOW,
TIMEinterval_min*60-1)
        # execute SQL query using execute() method.
        cursor.execute(sql)
        # Fetch a single row using fetchone() method.
        data = cursor.fetchone()
        while data is not None:
            if printinfo:
                #format time and rssi for print exit:
                hour = printformat(data[1].hour)
                minute = printformat(data[1].minute)
                second = printformat(data[1].second)
                rssi = printformat(data[0])

            tdiff = tnow - data[1] #find time difference in seconds
            TIMEtrustability = trust(tdiff.seconds,TIMEinterval_min)

            if printinfo:
                print
                rssi,"---
",hour+": "+minute+" "+second," \t",tdiff.seconds," \t",TIMEtrustability

            RSSIs_n_TIMEtrust.append((float(data[0]),TIMEtrustability))

```

```

        data = cursor.fetchone()

    if RSSIs_n_TIMEtrust:
        wRSSIs.append(wavgRSSI(RSSIs_n_TIMEtrust)) #weighted average
    else:
        wRSSIs.append(-1.0) #NO DATA

if printinfo:
    print "\nAverage RSSIs: "
    for i in range(len(meshnames)):
        print meshnames[i]+":",wRSSIs[i]

# disconnect from server
db.close()

##### MULTILATERATION #####

#De Rotterdam building geometry:
with fiona.open('./shp/base_polygon.shp', 'r') as c:
    building = c.next()['geometry']['coordinates']
c.close()
b = g.Polygon(building[0],building[1:])
Cpoint = g.Point(b.centroid)

#calculate rings:
rings = []
for i in range(len(wRSSIs)): #for each timeframe that we have 4(?) readings
    if wRSSIs[i] > 0:
        Spoint = g.Point(MeshPositions[i])
        dist = Spoint.distance(Cpoint)
        R = localize(wRSSIs[i],dist) #find radius
        Rerr = calcerr(wRSSIs[i]) #find error
        Couter = Spoint.buffer(R+Rerr)
        #!!!
        if R-Rerr > 0:
            Cinner = Spoint.buffer(R-Rerr)
        else:
            Cinner = Spoint.buffer(0.01)
        ring = Couter.difference(Cinner)
        #TODO: Indicator value could be anything as long as it is higher
        # when error is lower - but without having great differences!
        indicator = pow(1.0/(Rerr*len(meshnames)),2)
        rings.append( [ring, indicator] )
    else:
        rings.append(None)

#"cut out" the parts of the rings outside of the building:
leftoverings = []
for i in range(len(rings)):
    if rings[i]:
        inter = rings[i][0].intersection(b)
        if not inter.is_empty:
            if inter.type == "MultiPolygon":
                for interpoly in inter:
                    leftoverings.append([interpoly,rings[i][1]])
            else:
                leftoverings.append([inter,rings[i][1]])

#If shapefile representation is needed:
if saveshp:
    representation = represent(b,MeshPositions,leftoverings)
    if not representation:
        print "Unable to represent geometries. System Exit..."
        sys.exit()

#final intersection and area a person is localized
position = intersections.main(leftoverings,saveshp)
if position:
    return position[0]
else:
    return None

```

```

if __name__ == "__main__":
    print "\nExample\n\n"
    #meshlium table names from database
    meshnames = ['mesh121','mesh309','mesh553','mesh678']
    #Layout 2 meshlium scanner positions:
    MeshPositions = [(8.4573611030943816, 15.37890129341085732),
                    (32.19934627506354019, 14.53278265246893142),
                    (34.25747742550623798, 31.84169907549004463),
                    (8.56883399923329137, 28.68614385891768848) ]
    #cellphone MAC addresses to search
    #TODO: This will change in app to either the user's MAC or the MAC the user is trying to
find
    cellMAC = 'f8:e0:79:2f:02:45'
    #macs to test: 'f8:e0:79:2f:02:45','f8:e0:79:c1:9f:da','f8:e0:79:30:1b:87'
    TIMEinterval_min = 5.0 #in minutes
    #TODO: THIS WILL CHANGE IN THE SCRIPT to "now()" time in DB.
    timeNOW = '2014-10-01 17:25:00'
    main(meshnames,MeshPositions,cellMAC,timeNOW,TIMEinterval_min,True,True)

```

### Multilateration goback.py [similar to previous; cuts past data if differences above threshold]

```

import MySQLdb #download @ http://www.codegood.com/archives/129
from math import log, e
import sys
import shapely.geometry as g
import fiona
import intersections

#Weighted average based on time:
def wavgrssi(lstdata):
    return sum([v[0]*v[1] for v in lstdata]) / sum([v[1] for v in lstdata])

#Trustability based on how "old" the results are:
def trust(tdSEC,tinterMIN):
    if tinterMIN > 30:
        p = 1
    else:
        p = 15.0 / pow(log(tinterMIN+1,e),2)
    trustability = 1.0 - pow(log(tdSEC+1,tinterMIN*60+1),p) #+1 to avoid log(0)
    return trustability

#Localization radius based on translating RSSI values into distance
def localize(rssi, distFromCentr):
    Ro = 60.0 #RSSI at 1m distance
    R = float(rssi)
    #note: the thought was to parametrize the no value, based on the distance
    # of the scanner to the center of the building. Outcome were values
    # that didn't vary much from a static no value.
    #no = 3.62 - log(distFromCentr,Ro)
    no = 2.8
    n = no + log(Ro/R,Ro/2) #path-loss exponent
    x = (Ro-R) / (10*n)
    d = pow(10,x)
    return d

#Error function - defines thickness of distance localization rings
def calcerr(rssi):
    #Ro = 60.0 #dBm at 1m distance
    #err = log(Ro/float(rssi),2)
    #Error could be the above - instead, consider 2.0dBm differences
    err = abs(localize(rssi,0) - localize(rssi+2.0,0))
    return err

#Not really necessary, but printable info per single run
def printformat(fin):
    fin = int(fin)
    if fin < 10:
        fout = '0{0}'.format(fin)
    else:
        fout = '{0}'.format(fin)
    return fout

```



```

#Needed only in single run
def represent(building,MeshPositions,leftoverings):
    #REPRESENTATION: (.shp)
    # Define a polygon feature geometry - general schema
    polyschema = {
        'geometry': 'Polygon',
        'properties': {'id': 'int'},
    }
    pointschema = {
        'geometry': 'Point',
        'properties': {'id': 'int'},
    }
    #REPRESENT BUILDING:
    with fiona.open('derot.shp', 'w', 'ESRI Shapefile', polyschema) as c:
        c.write({
            'geometry': g.mapping(building),
            'properties': {'id': 0},
        })
    c.close()
    #REPRESENT MESHLIUM SCANNER POSITIONS:
    with fiona.open('MeshPositions.shp', 'w', 'ESRI Shapefile', pointschema) as c:
        for i in range(len(MeshPositions)):
            c.write({
                'geometry': g.mapping(g.Point(MeshPositions[i])),
                'properties': {'id': i},
            })
    c.close()
    #REPRESENT RINGS:
    with fiona.open('timeboxZ.shp', 'w', 'ESRI Shapefile', polyschema) as c:
        for i in range(len(leftoverings)):
            if leftoverings[i]:
                c.write({
                    'geometry': g.mapping(leftoverings[i][0]),
                    'properties': {'id': i},
                })
    c.close()
    return True

# MAIN FUNCTION
def main(meshnames,MeshPositions,cellMAC,timeNOW,TIMEinterval_min,saveshp,printinfo):

    if TIMEinterval_min <= 0:
        print "Time interval cannot be 0 or less"
        sys.exit()

    thres = 8.0 #threshold of variability in RSSI values

    # Open EXTERNAL database connection
    db = MySQLdb.connect(host="libelium.tudelft.nl",user="meshlium",
                        passwd="liuliu",db="meshliumdb" )
    # prepare a cursor object
    cursor = db.cursor()

    if printinfo:
        print "\n\n*** CELLPHONE MAC: ",cellMAC,"\nTIME NOW: ", timeNOW, "\nTIME TO SEARCH:
{0} MINUTES AGO".format(TIMEinterval_min)
    wRSSIs = [] # will contain weighted RSSI values per scanner
    for mesh in meshnames:
        RSSIs_n_TIMEtrust = []
        if printinfo:
            print "\n",mesh
    #TODO:time "now" in datetime format (POSSIBLY DELETE NEXT 3 LINES LATER)
    tsql = "SELECT timestamp('{0}')" .format(timeNOW)
    cursor.execute(tsql)
    tnow = cursor.fetchone()[0]

    #SQL query to get the RSSI values and datetimes
    sql = "" "SELECT RSSI, timestamp
FROM test.{0}
WHERE MAC='{1}' AND RSSI > 4
AND (timestamp between '{2}' - interval {3} second and '{2}')"
ORDER BY timestamp DESC"" .format(mesh, cellMAC, timeNOW,
TIMEinterval_min*60-1)

```

```

# execute SQL query using execute() method.
cursor.execute(sql)
# Fetch a single row using fetchone() method.
data = cursor.fetchone()
if data is not None:
    Ttemp = data[1]
    Rtemp = int(data[0])
    if (tnow - data[1]).seconds > 300:
        data = None
mdata = []
while data is not None:
    tdiff = tnow - data[1] #find time difference in seconds
    mdata.append( [float(data[0]), data[1], tdiff.seconds] )
    data = cursor.fetchone()
    if data is not None:
        if (abs(Rtemp - int(data[0])) > thres) or (Ttemp - data[1]).seconds > 300:
            data = None
        else:
            Ttemp = data[1]
            Rtemp = int(data[0])

for i in range(len(mdata)):
    TIMEtrustability = trust(mdata[i][2],float(mdata[-1][2]) / 60.0 + 1)
    if printinfo:
        #Format time and rssi for print exit:
        hour = printformat(mdata[i][1].hour)
        minute = printformat(mdata[i][1].minute)
        second = printformat(mdata[i][1].second)
        rssi = printformat(mdata[i][0])
        print
        ",hour+": "+minute+"."+second,"\\t",mdata[i][2],"\\t",TIMEtrustability
        RSSIs_n_TIMEtrust.append((float(mdata[i][0]),TIMEtrustability))

    if RSSIs_n_TIMEtrust:
        wRSSIs.append(wavg(rssi(RSSIs_n_TIMEtrust)) #weighted average
    else:
        wRSSIs.append(-1.0) #NO DATA

if printinfo:
    print "\\nAverage RSSIs: "
    for i in range(len(meshnames)):
        print meshnames[i]+":",wRSSIs[i]

# disconnect from server
db.close()

##### MULTILATERATION #####

#De Rotterdam building geometry:
with fiona.open('./shp/base_polygon.shp', 'r') as c:
    building = c.next()['geometry']['coordinates']
c.close()
b = g.Polygon(building[0],building[1:])
Cpoint = g.Point(b.centroid)

#calculate rings:
rings = []
for i in range(len(wRSSIs)): #for each timeframe that we have 4(?) readings
    if wRSSIs[i] > 0:
        Spoint = g.Point(MeshPositions[i])
        dist = Spoint.distance(Cpoint)
        R = localize(wRSSIs[i],dist) #find radius
        Rerr = calcerr(wRSSIs[i]) #find error
        Couter = Spoint.buffer(R+Rerr)
        #!!!
        if R-Rerr > 0:
            Cinner = Spoint.buffer(R-Rerr)
        else:
            Cinner = Spoint.buffer(0.01)
        ring = Couter.difference(Cinner)
        #TODO: Indicator value could be anything as long as it is higher
        # when error is lower - but without having great differences!

```

```

        indicator = pow(1.0/(Rerr*len(meshnames)),2)
        rings.append( [ring, indicator] )
    else:
        rings.append(None)

#"cut out" the parts of the rings outside of the building:
leftoverings = []
for i in range(len(rings)):
    if rings[i]:
        inter = rings[i][0].intersection(b)
        if not inter.is_empty:
            if inter.type == "MultiPolygon":
                for interpoly in inter:
                    leftoverings.append([interpoly,rings[i][1]])
            else:
                leftoverings.append([inter,rings[i][1]])

#If shapefile representation is needed:
if saveshp:
    representation = represent(b,MeshPositions,leftoverings)
    if not representation:
        print "Unable to represent geometries. System Exit..."
        sys.exit()

#final intersection and area a person is localized
position = intersections.main(leftoverings,saveshp)
if position:
    return position[0]
else:
    return None

if __name__ == "__main__":
    print "\nExample\n\n"
    #meshlium table names from database
    meshnames = ['mesh121','mesh309','mesh553','mesh678']
    #TEST 2 meshlium scanner positions:
    MeshPositions = [(8.4573611030943816, 15.37890129341085732),
                    (32.19934627506354019, 14.53278265246893142),
                    (34.25747742550623798, 31.84169907549004463),
                    (8.56883399923329137, 28.68614385891768848) ]
    #cellphone MAC addresses to search
    #TODO: This will change in app to either the user's MAC or the MAC the user is trying to
    find
    cellMAC = 'f8:e0:79:c1:9f:da'
    TIMEinterval_min = 60.0 #in minutes
    #TODO: THIS WILL CHANGE IN THE SCRIPT to "now()" time in DB.
    timeNOW = '2014-09-30 15:40:00'
    main(meshnames,MeshPositions,cellMAC,timeNOW,TIMEinterval_min,True,True)

```

### intersections.py [finds all possible areas for rings and intersections, and returns best solution]

```

from shapely import geometry as g
import fiona

def findintersections(polylst):
    if not polylst:
        return [],[]
    out = []
    full =[]
    for j in range(len(polylst)-1):
        intersected = False #a flag to check if polygon previously intersected
        #check if previously intersected:
        for i in range(j):
            if polylst[j][0].intersects(polylst[i][0]):
                intersected = True
                break
        #find intersections
        for i in range(j+1,len(polylst)):
            if polylst[j][0].intersects(polylst[i][0]):
                intersected = True
                inter = polylst[j][0].intersection(polylst[i][0])
                diff = polylst[j][0].difference(polylst[i][0])

```

```

        val = polylst[j][1] + polylst[i][1]
        #If difference area too small, the polygons are about the same
        if diff.area > 1.0:
            if inter.type == "MultiPolygon":
                for poly in inter:
                    out.append([poly,val])
            else:
                out.append([inter,val])
        else:
            intersected = False
            if intersected is False:
                full.append(polylst[j])
#check for last one if already intersected, otherwise add it:
            intersected = False
            for i in range(len(polylst)-1):
                if polylst[len(polylst)-1][0].intersects(polylst[i][0]):
                    intersected = True
                    break
            if intersected is False:
                full.append(polylst[len(polylst)-1])
            return [out,full]

def main(rings, representmax):
    if not rings:
        return None

    ringoutcome = []

    temp = findintersections(rings)
    ringoutcome.extend(temp[1]) #add the outcome unintersected polygons

    while temp[0]:
        temp = findintersections(temp[0])
        ringoutcome.extend(temp[1])

    #SCHEMA
    polyschema = {
        'geometry': 'Polygon',
        'properties': {'id': 'int'},
    }

    #Find best polygon, based on metric
    indx = -1
    maxvalue = -1
    for i in range(len(ringoutcome)):
        if ringoutcome[i][1] >= maxvalue:
            indx = i
            maxvalue = ringoutcome[i][1]

    #Depended on if representation is needed:
    if representmax:
        with fiona.open('foundmax.shp', 'w', 'ESRI Shapefile', polyschema) as c:
            c.write({
                'geometry': g.mapping(ringoutcome[indx][0]),
                'properties': {'id': int(maxvalue*1000)}#just to have an id
            })
        c.close()

    return ringoutcome[indx]

if __name__ == "__main__":
    #READ/test/random metric
    from random import randint
    c = fiona.open('timeboxZ.shp','r')
    rings = []
    for i in range(c.__len__()):
        pol = c.next()
        rings.append( [g.shape(pol['geometry']), randint(1,100)] )
    c.close()
    geom = main(rings,True)

```

### findroom.py [finds the best room-fit for solution area, based on subdivisions]

```
from shapely import geometry as g

#Finds the room identifier of localization
def main(locPOLY,roomlst):

    intersectionROOMS = []
    for room in roomlst:
        roomshape = g.Polygon(room[0][:-1])
        inter = locPOLY.intersection(roomshape)
        if not inter.is_empty:
            intersectionROOMS.append((inter.area,room[1]))

    maxarea = -1
    roomID = -1
    for inter in intersectionROOMS:
        if inter[0] > maxarea:
            maxarea = inter[0]
            roomID = inter[1]

    return roomID

if __name__ == "__main__":
    print "Utility function for runner.py"
```

### priority.py [helpful function for 'complex' runner that returns an ordered list of rooms]

```
from shapely import geometry as g

#Returns a list of ids of the intuitive subdivision rooms which intersect
#better per each automatic subdivision area:
def main(rooms, extrarooms):
    lst = []
    for extra in extrarooms:
        templst = []
        for room in rooms:
            if g.Polygon(extra[0]).intersects(g.Polygon(room[0])) and \
                not (g.Polygon(extra[0]).touches(g.Polygon(room[0]))):
                templst.append( (g.Polygon(extra[0]).intersection(g.Polygon(room[0])).area,
room[1]) )
        templst.sort() #sort by area (ASC)
        templst.reverse() #reverse to have sorted list DESC
        lst.append(zip(*templst)[1]) #only store ids
    return lst

if __name__ == "__main__":
    print "Utility function for runner_complex"
```

### runner SUBDIVISION intuitive.py [tester for intuitive subdivision]

```
# script to run for all points - INTUITIVE SUBDIVISION
#import multilateration_goback
import multilateration
import findroom
import fiona
import sys

#meshlium table names from database #SPECIFIC ORDER
meshnames = ['mesh121','mesh309','mesh553','mesh678']

#cellphones MAC addresses to search
#TODO: This will change in app to either the user's MAC or the MAC the user is trying to find
cellMACs = ['f8:e0:79:2f:02:45','f8:e0:79:c1:9f:da','f8:e0:79:30:1b:87']

#TODO: Need to find a way to choose which interval is trustable enough?
TIMEinterval_min = 5.0 #in minutes (for testing)
#TIMEinterval_min = 60.0 #in minutes (for goback)
```

```

#Space subdivision- get from shapefile:
rooms = []
with fiona.open('./shp/intuitive_subdivision.shp', 'r') as c:
    for poly in c:
        coords = []
        for xy in poly['geometry']['coordinates'][0]:
            coords.append( (round(xy[0],6),round(xy[1],6)) )
        rooms.append( (coords,int(poly['properties']['FID_'])) )
c.close()

#SUBDIVISION 1 (logical/human areas)
#Dependent on space subdivision:
neighs = [ [1,6], [0,2], [1,3], [2,4,5,7], [3], [3,6], [0,5,7], [3,6] ]
lay = raw_input("Choose Layout [1,2,3 or 4]:\n")
if lay == '1' or lay == '2' or lay == '3':
    #true point tested (id of room for each point)
    truePOS = [ 0, 0, 1, 1, 2, 2, 3, 7, 6, 6, 6, 5, 5, 3, 4, 4, 1, 1, 2, 2, 5, 5]

##### LAYOUT 1 #####
if lay == '1':
    tes = raw_input("Choose Test [1 for 30/09, 2 for 01/10]:\n")
    #Point-like positions of meshliums scanners #SPECIFIC ORDER
    MeshPositions = [(2.73631731473008211, 2.29640446220620564),
                    (40.26027550729227045, 9.82033913232249134),
                    (40.26027550729227755, 43.82144896367393017),
                    (2.73631731473008566, 33.77127430523957941) ]

    if tes == '1':
        #30/09
        timesNOW = ['2014-09-30 12:45:00', '2014-09-30 12:50:00', '2014-09-30 12:55:00',
                    '2014-09-30 13:00:00', '2014-09-30 13:05:00', '2014-09-30 13:10:00',
                    '2014-09-30 13:15:00', '2014-09-30 13:20:00', '2014-09-30 13:25:00',
                    '2014-09-30 13:30:00', '2014-09-30 13:35:00', '2014-09-30 13:40:00',
                    '2014-09-30 13:45:00', '2014-09-30 13:50:00', '2014-09-30 14:00:00',
                    '2014-09-30 14:05:00', '2014-09-30 14:15:00', '2014-09-30 14:20:00',
                    '2014-09-30 14:25:00', '2014-09-30 14:30:00', '2014-09-30 14:35:00',
                    '2014-09-30 14:40:00']

    elif tes == '2':
        #01/10
        timesNOW = ['2014-10-01 14:25:00', '2014-10-01 14:30:00', '2014-10-01 14:35:00',
                    '2014-10-01 14:40:00', '2014-10-01 14:45:00', '2014-10-01 14:50:00',
                    '2014-10-01 14:55:00', '2014-10-01 15:00:00', '2014-10-01 15:05:00',
                    '2014-10-01 15:10:00', '2014-10-01 15:15:00', '2014-10-01 15:20:00',
                    '2014-10-01 15:25:00', '2014-10-01 15:30:00', '2014-10-01 15:35:00',
                    '2014-10-01 15:40:00', '2014-10-01 15:45:00', '2014-10-01 15:50:00',
                    '2014-10-01 15:55:00', '2014-10-01 16:00:00', '2014-10-01 16:05:00',
                    '2014-10-01 16:10:00']

    else:
        print "Wrong Choice!"
        sys.exit()

##### LAYOUT 2 #####
elif lay == '2':
    tes = raw_input("Choose Test [1 for 30/09, 2 for 01/10]:\n")
    MeshPositions = [(8.4573611030943816, 15.37890129341085732),
                    (32.19934627506354019, 14.53278265246893142),
                    (34.25747742550623798, 31.84169907549004463),
                    (8.56883399923329137, 28.68614385891768848) ]

    if tes == '1':
        #30/09
        timesNOW = ['2014-09-30 15:15:00', '2014-09-30 15:20:00', '2014-09-30 15:25:00',
                    '2014-09-30 15:30:00', '2014-09-30 15:35:00', '2014-09-30 15:40:00',
                    '2014-09-30 15:45:00', '2014-09-30 15:50:00', '2014-09-30 15:55:00',
                    '2014-09-30 16:00:00', '2014-09-30 16:05:00', '2014-09-30 16:10:00',
                    '2014-09-30 16:15:00', '2014-09-30 16:20:00', '2014-09-30 16:25:00',
                    '2014-09-30 16:30:00', '2014-09-30 16:35:00', '2014-09-30 16:40:00',
                    '2014-09-30 16:45:00', '2014-09-30 16:50:00', '2014-09-30 16:55:00',
                    '2014-09-30 17:00:00']

    elif tes == '2':
        #01/10
        timesNOW = ['2014-10-01 16:30:00', '2014-10-01 16:35:00', '2014-10-01 16:40:00',
                    '2014-10-01 16:45:00', '2014-10-01 16:50:00', '2014-10-01 16:55:00',
                    '2014-10-01 17:00:00', '2014-10-01 17:05:00', '2014-10-01 17:10:00',
                    '2014-10-01 17:15:00', '2014-10-01 17:20:00', '2014-10-01 17:25:00',
                    '2014-10-01 17:30:00', '2014-10-01 17:35:00', '2014-10-01 17:40:00',
                    '2014-10-01 17:45:00', '2014-10-01 17:52:00', '2014-10-01 17:57:00',

```

```

        '2014-10-01 18:02:00', '2014-10-01 18:07:00', '2014-10-01 18:12:00',
        '2014-10-01 18:17:00']
    else:
        print "Wrong Choice!"
        sys.exit()
##### LAYOUT 3 #####
elif lay == '3':
    MeshPositions = [(4.52033820207128745, 20.23523715507049303),
                    (20.89279016717234683, 10.99609148383166968),
                    (39.96342015864106401, 21.73612628962893112),
                    (20.89279016717235393, 33.41814505422586024) ]
    timesNOW = ['2014-10-01 10:10:00', '2014-10-01 10:15:00', '2014-10-01 10:20:00',
                '2014-10-01 10:25:00', '2014-10-01 10:30:00', '2014-10-01 10:35:00',
                '2014-10-01 10:40:00', '2014-10-01 10:45:00', '2014-10-01 10:50:00',
                '2014-10-01 10:55:00', '2014-10-01 11:00:00', '2014-10-01 11:05:00',
                '2014-10-01 11:10:00', '2014-10-01 11:15:00', '2014-10-01 11:25:00',
                '2014-10-01 11:30:00', '2014-10-01 11:37:00', '2014-10-01 11:42:00',
                '2014-10-01 11:48:00', '2014-10-01 11:53:00', '2014-10-01 11:59:00',
                '2014-10-01 12:04:00']
##### LAYOUT 4 #####
elif lay == '4':
    MeshPositions = [(19.00734526019633108, 10.16990537971109987),
                    (40.26027550729227045, 9.82033913232249134),
                    (40.26027550729227755, 43.82144896367393017),
                    (19.00734526019633108, 33.68398416078886015) ]
    #true point tested (id of room for each point) / different for layout 4
    truePOS = [ 3, 3, 7, 6, 6, 5, 5, 4, 4, 5, 5]
    timesNOW = ['2014-10-01 12:50:00', '2014-10-01 12:55:00', '2014-10-01 13:00:00',
                '2014-10-01 13:05:00', '2014-10-01 13:10:00', '2014-10-01 13:15:00',
                '2014-10-01 13:20:00', '2014-10-01 13:25:00', '2014-10-01 13:30:00',
                '2014-10-01 13:35:00', '2014-10-01 13:41:00', '2014-10-01 13:46:00']
else:
    print "Wrong Choice!"
    sys.exit()

#RUNNER - for the above choices
for cell in cellMACs:
    room = []
    on = 0
    close = 0
    for t in range(len(timesNOW)):
        pos = multilateration.main(meshnames,MeshPositions,cell,timesNOW[t],
                                  TIMEinterval_min,False,False)

        if pos:
            roomID = findroom.main(pos,rooms)
        else:
            roomID = -1
        if roomID == truePOS[t]:
            on += 1
            close += 1
        room.append(roomID)
        if roomID in neighs[truePOS[t]]:
            close += 1

    print "\n\n~~~~~"
    print "~~~ End of operations for cellphone:", cell, "~~~"
    print "~~~~~"
    print "Point numbers on
plan:|01|02|03|04|05|06|07|08|09|10|11|12|13|14|15|16|17|18|19|20|21|22|"
    print "Ground Truth room IDs:",truePOS
    print "Localization room IDs:",room
    print "Success Rate:",on*100.0 / len(truePOS),"%", "({0} out of {1} times correctly
localized)".format(on, len(truePOS))
    print "Extend to Neighbours Rate:",close*100.0 / len(truePOS),"%"

```

### runner SUBDIVISION automatic layout1.py [tester for automatic subdivision]

```

# script to run for all points - Automatic Subdivision - ONLY LAYOUT 1
#import multilateration_goback
import multilateration

```

```

import findroom
import fiona
import sys

#meshlium table names from database #SPECIFIC ORDER
meshnames = ['mesh121','mesh309','mesh553','mesh678']

#cellphones MAC addresses to search
#TODO: This will change in app to either the user's MAC or the MAC the user is trying to find
cellMACs = ['f8:e0:79:2f:02:45','f8:e0:79:c1:9f:da','f8:e0:79:30:1b:87']

#TODO: Need to find a way to choose which interval is trustable enough?
TIMEinterval_min = 5.0 #in minutes

#Space subdivision- get from shapefile:
rooms = []
with fiona.open('./shp/automatic_subdivision.shp', 'r') as c:
    for poly in c:
        coords = []
        for xy in poly['geometry']['coordinates'][0]:
            coords.append( (round(xy[0],6),round(xy[1],6)) )
        rooms.append( (coords,int(poly['properties']['FID'])) )
c.close()

#SUBDIVISION 1 (automatic/scanner-reach areas)
#Dependent on space subdivision:
neighs = [ [1,7], [0,6,7], [4,7], [4,5], [2,3,5,6,7], [3,4,6], [1,4,5,7], [0,1,2,4,6] ]
print "Only layout 1 applicable."
#true point tested (id of room for each point)
truePOS = [ 0, 7, 7, 7, 1, 1, 6, 6, 7, 2, 2, 2, 4, 5, 3, 3, 1, 0, 1, 1, 2, 2]

##### LAYOUT 1 #####

tes = raw_input("Choose Test [1 for 30/09, 2 for 01/10]:\n")
#Point-like positions of meshliums scanners #SPECIFIC ORDER
MeshPositions = [(2.73631731473008211, 2.29640446220620564),
(40.26027550729227045, 9.82033913232249134),
(40.26027550729227755, 43.82144896367393017),
(2.73631731473008566, 33.77127430523957941) ]

if tes == '1':
    #30/09
    timesNOW = ['2014-09-30 12:45:00','2014-09-30 12:50:00','2014-09-30 12:55:00',
'2014-09-30 13:00:00','2014-09-30 13:05:00','2014-09-30 13:10:00',
'2014-09-30 13:15:00','2014-09-30 13:20:00','2014-09-30 13:25:00',
'2014-09-30 13:30:00','2014-09-30 13:35:00','2014-09-30 13:40:00',
'2014-09-30 13:45:00','2014-09-30 13:50:00','2014-09-30 14:00:00',
'2014-09-30 14:05:00','2014-09-30 14:15:00','2014-09-30 14:20:00',
'2014-09-30 14:25:00','2014-09-30 14:30:00','2014-09-30 14:35:00',
'2014-09-30 14:40:00']
elif tes == '2':
    #01/10
    timesNOW = ['2014-10-01 14:25:00','2014-10-01 14:30:00','2014-10-01 14:35:00',
'2014-10-01 14:40:00','2014-10-01 14:45:00','2014-10-01 14:50:00',
'2014-10-01 14:55:00','2014-10-01 15:00:00','2014-10-01 15:05:00',
'2014-10-01 15:10:00','2014-10-01 15:15:00','2014-10-01 15:20:00',
'2014-10-01 15:25:00','2014-10-01 15:30:00','2014-10-01 15:35:00',
'2014-10-01 15:40:00','2014-10-01 15:45:00','2014-10-01 15:50:00',
'2014-10-01 15:55:00','2014-10-01 16:00:00','2014-10-01 16:05:00',
'2014-10-01 16:10:00']
else:
    print "Wrong Choice!"
    sys.exit()

#RUNNER - for the above choices
for cell in cellMACs:
    room = []
    on = 0
    close = 0
    for t in range(len(timesNOW)):
        pos = multilateration.main(meshnames,MeshPositions,cell,timesNOW[t],
TIMEinterval_min,False,False)

        if pos:
            roomID = findroom.main(pos,rooms)
        else:

```



```

        roomID = -1
    if roomID == truePOS[t]:
        on += 1
        close += 1
    room.append(roomID)
    if roomID in neighs[truePOS[t]]:
        close += 1

    print "\n\n~~~~~"
    print "~~~ End of operations for cellphone:", cell,"~~~"
    print "~~~~~"
    print "Point numbers on
plan: |01|02|03|04|05|06|07|08|09|10|11|12|13|14|15|16|17|18|19|20|21|22|"
    print "Ground Truth room IDs:",truePOS
    print "Localization room IDs:",room
    print "Success Rate:",on*100.0 / len(truePOS),"%", "({0} out of {1} times correctly
localized)".format(on, len(truePOS))
    print "Extend to Neighbours Rate:",close*100.0 / len(truePOS),"%"

```

### runner SUBDIVISION complex.py [tester for combined method of intuitive+automatic]

```

# script to run for all points - uses both intuitive (basis) and automatic
# subdivisions - ONLY LAYOUT 1
#import multilateration_goback
import multilateration
import findroom
import fiona
import sys
import priority

#meshlium table names from database #SPECIFIC ORDER
meshnames = ['mesh121','mesh309','mesh553','mesh678']

#cellphones MAC addresses to search
#TODO: This will change in app to either the user's MAC or the MAC the user is trying to find
cellMACs = ['f8:e0:79:2f:02:45','f8:e0:79:c1:9f:da','f8:e0:79:30:1b:87']

#TODO: Need to find a way to choose which interval is trustable enough?
TIMEinterval_min = 5.0 #in minutes

#Space subdivision- get from shapefile:
rooms = []
with fiona.open('./shp/intuitive_subdivision.shp', 'r') as c:
    for poly in c:
        coords =[]
        for xy in poly['geometry']['coordinates'][0]:
            coords.append( (round(xy[0],6),round(xy[1],6)) )
        rooms.append( (coords,int(poly['properties']['FID_'])) )
c.close()

#Utility subdivision for next possible room based on scanner placement:
extrarooms = []
with fiona.open('./shp/automatic_subdivision.shp', 'r') as c:
    for poly in c:
        coords =[]
        for xy in poly['geometry']['coordinates'][0]:
            coords.append( (round(xy[0],6),round(xy[1],6)) )
        extrarooms.append( (coords,int(poly['properties']['FID_'])) )
c.close()

#Priority list of next possible room:
nextrooms = priority.main(rooms, extrarooms)

#SUBDIVISION 1 (logical/human areas)
#true point tested (id of room for each point)
truePOS = [ 0, 0, 1, 1, 2, 2, 3, 7, 6, 6, 6, 5, 5, 3, 4, 4, 1, 1, 2, 2, 5, 5]

##### LAYOUT 1 #####

tes = raw_input("Choose Test [1 for 30/09, 2 for 01/10]:\n")
#Point-like positions of meshliums scanners #SPECIFIC ORDER

```

```

MeshPositions = [(2.73631731473008211, 2.29640446220620564),
                 (40.26027550729227045, 9.82033913232249134),
                 (40.26027550729227755, 43.82144896367393017),
                 (2.73631731473008566, 33.77127430523957941) ]

if tes == '1':
    #30/09
    timesNOW = ['2014-09-30 12:45:00', '2014-09-30 12:50:00', '2014-09-30 12:55:00',
                '2014-09-30 13:00:00', '2014-09-30 13:05:00', '2014-09-30 13:10:00',
                '2014-09-30 13:15:00', '2014-09-30 13:20:00', '2014-09-30 13:25:00',
                '2014-09-30 13:30:00', '2014-09-30 13:35:00', '2014-09-30 13:40:00',
                '2014-09-30 13:45:00', '2014-09-30 13:50:00', '2014-09-30 14:00:00',
                '2014-09-30 14:05:00', '2014-09-30 14:15:00', '2014-09-30 14:20:00',
                '2014-09-30 14:25:00', '2014-09-30 14:30:00', '2014-09-30 14:35:00',
                '2014-09-30 14:40:00']

elif tes == '2':
    #01/10
    timesNOW = ['2014-10-01 14:25:00', '2014-10-01 14:30:00', '2014-10-01 14:35:00',
                '2014-10-01 14:40:00', '2014-10-01 14:45:00', '2014-10-01 14:50:00',
                '2014-10-01 14:55:00', '2014-10-01 15:00:00', '2014-10-01 15:05:00',
                '2014-10-01 15:10:00', '2014-10-01 15:15:00', '2014-10-01 15:20:00',
                '2014-10-01 15:25:00', '2014-10-01 15:30:00', '2014-10-01 15:35:00',
                '2014-10-01 15:40:00', '2014-10-01 15:45:00', '2014-10-01 15:50:00',
                '2014-10-01 15:55:00', '2014-10-01 16:00:00', '2014-10-01 16:05:00',
                '2014-10-01 16:10:00']

else:
    print "Wrong Choice!"
    sys.exit()

#RUNNER - for the above choices
for cell in cellMACs:
    room = []
    nextroom = []
    on = 0
    adj = 0
    for t in range(len(timesNOW)):
        pos = multilateration.main(meshnames, MeshPositions, cell, timesNOW[t],
                                   TIMEinterval_min, False, False)

        if pos:
            roomID = findroom.main(pos, rooms)
            autoID = findroom.main(pos, extrarooms)
        else:
            roomID = -1
            autoID = -1
        room.append(roomID)
        if autoID != -1:
            for nextID in nextrooms[autoID]:
                if nextID != roomID:
                    nextroom.append(nextID)
                    break
        else:
            nextroom.append(-1)

        #counters:
        if roomID == truePOS[t]:
            on += 1
            adj += 1
        elif nextID == truePOS[t]:
            adj += 1

    print "\n\n~~~~~"
    print "~~~ End of operations for cellphone:", cell, "~~~"
    print "~~~~~"
    print "Point numbers on
plan: |01|02|03|04|05|06|07|08|09|10|11|12|13|14|15|16|17|18|19|20|21|22|"
    print "Ground Truth room IDs:", truePOS
    print "Localization room IDs:", room
    print "Localization next IDs:", nextroom
    print "Success Rate:", on*100.0 / len(truePOS), "%", "({0} out of {1} times correctly
localized)".format(on, len(truePOS))
    print "Extend to Next Room Rate:", adj*100.0 / len(truePOS), "%", "({0} out of {1} times 2-
room localization)".format(adj, len(truePOS))

```

## APPENDIX 5. PYTHON 2.7 CODE: WIFI FINGERPRINTING SCRIPTS

### main.py

```
from read_data import read_data
from interpolation import interpolation
from draw_heatmap import draw_heatmap
from localization import localization
from read_polygon import read_polygon
from intersection import intersection
import pprint

# read the data from a fixed-format txt file
[n1,n2,rows,columns,width,ids,x,y,z1,z2,z3,z4]=read_data('input_L1_22.txt')

# interpolate the input points to create heatmaps
heatmaps=interpolation(n1,rows,columns,x,y,z1,z2,z3,z4)

# draw the heatmaps
#draw_heatmap(heatmaps,rows,columns)

# localize the testing points in the heatmaps
lcz=localization(n1,n2,rows,columns,x,y,z1,z2,z3,z4,heatmaps)
# print the localization result
pprint.pprint(lcz) #[x[n1+k],y[n1+k],xx,yy,dx,dy,2*math.sqrt(dx**2+dy**2),mindiff]

#intersect the localized cells with the subdivisions and write it into shapefile
grid_fp='Shapefiles\Grid22_ply.shp'
subs_fp0='Shapefiles\Final_space_subdivision.shp'
subs_fp1='Shapefiles\SubdivisionLayout1.shp'
subs0=read_polygon(subs_fp0)
subs1=read_polygon(subs_fp1)
cells=read_polygon(grid_fp)

#print the given subdivisions in priority order for each testing points
pprint.pprint(intersection(rows,columns,lcz,cells,subs0,subs1))
```

### read\_data.py

```
def read_data(input):
#read the data from a fixed format txt file
fh=open(input,"r")
lines=fh.readlines()
fh.close()
data=lines[0].strip().split(' ')

# number of rows in the grid
rows=int(data[0])
# number of columns in the grid
columns=int(data[1])
# width of the cell in the grid
width=int(data[2])
# number of sampling points
n1=int(lines[1])
# point id
ids=[]
# x coordinate of the points # x coordinate of the points
x=[]
# y coordinate of the points # x coordinate of the points
y=[]
# RSSI values from 1st scanner
z1=[]
# RSSI values from 2nd scanner
z2=[]
# RSSI values from 3rd scanner
z3=[]
# RSSI values from 4th scanner
z4=[]
for line in lines[2:n1+2]:
    data=line.strip().split(' ')
    ids.append(int(data[0]))
```

```

        x.append(int(data[1]))
        y.append(int(data[2]))
        z1.append(float(data[3]))
        z2.append(float(data[4]))
        z3.append(float(data[5]))
        z4.append(float(data[6]))

# number of testing points
n2=int(lines[n1+2])
for line in lines[n1+3:n1+n2+3]:
    data=line.strip().split(' ')
    ids.append(int(data[0]))
    x.append(int(data[1]))
    y.append(int(data[2]))
    z1.append(float(data[3]))
    z2.append(float(data[4]))
    z3.append(float(data[5]))
    z4.append(float(data[6]))

return [n1,n2,rows,columns,width,ids,x,y,z1,z2,z3,z4]

```

### interpolation.py

```

import numpy
import scipy.interpolate

def interpolation(n1,rows,columns,x,y,z1,z2,z3,z4):
# create heat maps for each scanner

# create a mesh grid
grid_x, grid_y = numpy.mgrid[0:rows, 0:columns]
# Interpolation
rbf1 = scipy.interpolate.Rbf(x[0:n1], y[0:n1], z1[0:n1], function='thin_plate')
Z1 = rbf1(grid_x, grid_y)
rbf2 = scipy.interpolate.Rbf(x[0:n1], y[0:n1], z2[0:n1], function='thin_plate')
Z2 = rbf2(grid_x, grid_y)
rbf3 = scipy.interpolate.Rbf(x[0:n1], y[0:n1], z3[0:n1], function='thin_plate')
Z3 = rbf3(grid_x, grid_y)
rbf4 = scipy.interpolate.Rbf(x[0:n1], y[0:n1], z4[0:n1], function='thin_plate')
Z4 = rbf4(grid_x, grid_y)

# make values below 0 to be 0
for i in range(0,rows):
    for j in range(0,columns):
        if (Z1[i][j]<0):
            Z1[i][j]=0
for i in range(0,rows):
    for j in range(0,columns):
        if (Z2[i][j]<0):
            Z2[i][j]=0
for i in range(0,rows):
    for j in range(0,columns):
        if (Z3[i][j]<0):
            Z3[i][j]=0
for i in range(0,rows):
    for j in range(0,columns):
        if (Z4[i][j]<0):
            Z4[i][j]=0

#exclude the part outside building (for 2 by 2 grid)
for i in range(0,5):
    for j in range(0,11):
        Z1[i][j]=float('nan')
        Z2[i][j]=float('nan')
        Z3[i][j]=float('nan')
        Z4[i][j]=float('nan')
for i in range(9,15):
    for j in range(4,8):
        Z1[i][j]=float('nan')
        Z2[i][j]=float('nan')
        Z3[i][j]=float('nan')
        Z4[i][j]=float('nan')
for k in range(12,17):
    Z1[i][k]=float('nan')

```

```

        Z2[i][k]=float('nan')
        Z3[i][k]=float('nan')
        Z4[i][k]=float('nan')
for i in range(19,23):
    for j in range(9,21):
        Z1[i][j]=float('nan')
        Z2[i][j]=float('nan')
        Z3[i][j]=float('nan')
        Z4[i][j]=float('nan')

#exclude the part outside building (for 4 by 4 grid)
# for i in range(0,3):
#     # for j in range(0,5):
#         # Z1[i][j]=float('nan')
#         # Z2[i][j]=float('nan')
#         # Z3[i][j]=float('nan')
#         # Z4[i][j]=float('nan')
#     # for i in range(5,8):
#         # for j in range(2,4):
#             # Z1[i][j]=float('nan')
#             # Z2[i][j]=float('nan')
#             # Z3[i][j]=float('nan')
#             # Z4[i][j]=float('nan')
#         # for k in range(6,8):
#             # Z1[i][k]=float('nan')
#             # Z2[i][k]=float('nan')
#             # Z3[i][k]=float('nan')
#             # Z4[i][k]=float('nan')
# for i in range(10,12):
#     # for j in range(4,11):
#         # Z1[i][j]=float('nan')
#         # Z2[i][j]=float('nan')
#         # Z3[i][j]=float('nan')
#         # Z4[i][j]=float('nan')
return [Z1,Z2,Z3,Z4]

```

### draw\_heatmap.py

```

import matplotlib.pyplot as plt

def draw_heatmap(heatmaps,rows,columns):
# Draw the heat maps
for i in range(0,len(heatmaps)):
    plt.subplot(2, len(heatmaps)/2, i+1)
    plt.imshow(heatmaps[i], extent=(1,columns,1,rows), origin='upper', vmin=0, vmax=60)
    plt.title('Scanner{0}'.format(i+1))
    plt.colorbar()
    plt.show()

```

### localization.py

```

import math

def localization(n1,n2,rows,columns,x,y,z1,z2,z3,z4,heatmaps):
# localize testing points in the grid using the heat maps created with the sampling points

lcz=[]
for k in range(0,n2):
    mindiff=9999;
    for i in range(0,rows):
        for j in range(0,columns):
            if math.isnan(heatmaps[0][i][j]):
                continue
            diff=math.sqrt((z1[n1+k]-heatmaps[0][i][j])**2+(z2[n1+k]-
heatmaps[1][i][j])**2+(z3[n1+k]-heatmaps[2][i][j])**2+(z4[n1+k]-
heatmaps[3][i][j])**2)
            if (diff<mindiff):
                xx=i
                yy=j
                mindiff=diff
        dx=x[n1+k]-xx
        dy=y[n1+k]-yy
        lcz.append([x[n1+k],y[n1+k],xx,yy,dx,dy,2*math.sqrt(dx**2+dy**2),mindiff])
return lcz

```

## intersection.py

```
from grid_intersect_sub import grid_intersect_sub

def intersection(lcz,cells,subs0,subs1):
# find the subdivision which the localized cell lies in through intersection
rec=[]
for i in range(0,len(lcz)):
    x0=lcz[i][0] #Ground truth x
    y0=lcz[i][1] #Ground truth y
    xx=lcz[i][2] #Calculated x
    yy=lcz[i][3] #Calculated y

N0=columns*(rows-x0-1)+y0 #Index of ground truth cell in the grid
N1=columns*(rows-xx-1)+yy #Index of calculated cell in the grid

#Index of ground truth subdivision
indx_gt=grid_intersect_sub(cells[N0],subs0)[-1][1]
#Index of the subdivision which the calculated cell lies in
indx_s0_1st=grid_intersect_sub(cells[N1],subs0)[-1][1]

    # check if it is correct. if not, a second or even a third subdivision alternative will
be given
    if indx_s0_1st != indx_gt:
        # Check in which sub of subs1 the calculated cell lies
        indx_s1=grid_intersect_sub(cells[N1],subs1)[-1][1]
        # Check which subs of subs0 intersect with this sub of subs1
        indx_s0s1=grid_intersect_sub(subs1[indx_s1],subs0)

        if indx_s0_1st==indx_s0s1[-1][1] and len(indx_s0s1)>=2:
            indx_s0_2rd=indx_s0s1[-2][1]
            if indx_s0_2rd != indx_gt and len(indx_s0s1)>=3:
                indx_s0_3rd=indx_s0s1[-3][1]
                if indx_s0_3rd==indx_gt:
                    result='localized successfully'
                else:
                    result='3 chances at most, localization failed'
            else:
                if indx_s0_2rd == indx_gt:
                    indx_s0_3rd=None
                    result='localized successfully'
                else:
                    indx_s0_3rd=None
                    result='no other choice, localization failed'

        elif indx_s0s1[-1][1] != indx_s0_1st and len(indx_s0s1)>=1:
            indx_s0_2rd=indx_s0s1[-1][1]
            if indx_s0_2rd != indx_gt and len(indx_s0s1)>=2:
                indx_s0_3rd=indx_s0s1[-2][1]
                if indx_s0_3rd==indx_s0_1st:
                    if len(indx_s0s1)>=3:
                        indx_s0_3rd=indx_s0s1[-3][1]
                        if indx_s0_3rd==indx_gt:
                            result='localized successfully'
                        else:
                            result='3 chances at most, localization failed'
                    else:
                        indx_s0_3rd=None
                        result='no other choice, localization failed'
                else:
                    if indx_s0_3rd==indx_gt:
                        result='localized successfully'
                    else:
                        result='3 chances at most, localization failed'
            else:
                if indx_s0_2rd == indx_gt:
                    indx_s0_3rd=None
                    result='localized successfully'
                else:
                    indx_s0_3rd=None
                    result='no other choice, localization failed'
        else:
            indx_s0_2rd=None
```

```

        indx_s0_3rd=None
        result='no other choice, localization failed'
    else:
        indx_s0_2rd=None
        indx_s0_3rd=None
        result='localized successfully'

rec.append([N0,N1,indx_gt,indx_s0_1st,indx_s0_2rd,indx_s0_3rd,result])
return rec

```

#### grid\_intersect\_sub.py

```

def grid_intersect_sub(cell,subs):
#intersect grid cells with subdivisions
rec=[]
for i in range(0,len(subs)):
    if cell.intersects(subs[i]):
        rec.append((cell.intersection(subs[i]).area, i))
rec.sort()
return rec

```

## APPENDIX 6. CODE FOR THE ANDROID APPLICATION

### MainScreenActivity.java

```

package com.example.derotterdamappsql;
public class MainScreenActivity extends Activity{
    Button btnViewDepartments;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_screen);
        // Button
        btnViewDepartments = (Button) findViewById(R.id.btnViewDepartments);
        // view products click event
        btnViewDepartments.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Launching All products Activity
                Intent i = new Intent(getApplicationContext(), AllDepartmentsActivity.class);
                startActivity(i);
            }
        });
    }
}

```

### AllDepartmentsActivity.java

```

package com.example.derotterdamappsql;
import com.example.derotterdamappsql.R;
public class AllDepartmentsActivity extends ListActivity {
    // Progress Dialog
    private ProgressDialog pDialog;
    // Creating JSON Parser object

```

```

JSONParser jParser = new JSONParser();
ArrayList<HashMap<String, String>> catchdepartmentsList;
// url to get all products list
private static String url_all_departments =
"http://server.kirupa.nl/damien/get_all_departments.php";
// JSON Node names
private static final String TAG_SUCCESS = "success";
private static final String TAG_CATCHDEPARTMENTS = "catchdepartments";
private static final String TAG_DEPID = "depid";
private static final String TAG_DEPARTMENT = "department";
// products JSONArray
JSONArray catchdepartments = null;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.all_departments);
    // Hashmap for ListView
    catchdepartmentsList = new ArrayList<HashMap<String, String>>();
    // Loading products in Background Thread
    new LoadAllDepartments().execute();
    // Get listview
    ListView lv = getListView();
    // on click launch All Phonse Activity
    lv.setOnItemClickListener(new OnItemClickListener() {
    @Override
        public void onItemClick(AdapterView<?> parent, View view,
            int position, long id) {
            // getting values from selected ListItem
            String depid = ((TextView) view.findViewById(R.id.depid)).getText()
                .toString();
            // Starting new intent
            Intent in = new Intent(getApplicationContext(),
                AllPhonesActivity.class);
            // sending phoneid to next activity
            in.putExtra(TAG_DEPID, depid);
            // starting new activity and expecting some response back
            startActivityForResult(in, 100);
        }
    });
}
// Response from All Phones Activity
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    // if result code 100
    if (resultCode == 100) {
        Intent intent = getIntent();
        finish();
        startActivity(intent);
    }
}

```



```

    }
}
/**
 * Background Async Task to Load all product by making HTTP Request
 */
class LoadAllDepartments extends AsyncTask<String, String, String> {
/**
 * Before starting background thread Show Progress Dialog
 */
@Override
protected void onPreExecute() {
    super.onPreExecute();
    pDialog = new ProgressDialog(AllDepartmentsActivity.this);
    pDialog.setMessage("Loading departments. Please wait...");
    pDialog.setIndeterminate(false);
    pDialog.setCancelable(false);
    pDialog.show();
}
/**
 * getting All phones from URL
 */
protected String doInBackground(String... args) {
    // Building Parameters
    List<NameValuePair> params = new ArrayList<NameValuePair>();
    // getting JSON string from URL
    JSONObject json = jParser.makeHttpRequest(url_all_departments, "GET",
params);
    // Check your log cat for JSON reponse
    Log.d("All Departments: ", json.toString());
    try {
        // Checking for SUCCESS TAG
        int success = json.getInt(TAG_SUCCESS);
        if (success == 1) {
            // products found
            // Getting Array of Phones
            catchdepartments = json.getJSONArray(TAG_CATCHDEPARTMENTS);
            // looping through All Products
            for (int i = 0; i < catchdepartments.length(); i++) {
                JSONObject c = catchdepartments.getJSONObject(i);
                // Storing each json item in variable
                String depid = c.getString(TAG_DEPID);
                String department = c.getString(TAG_DEPARTMENT);
                // creating new HashMap
                HashMap<String, String> map = new HashMap<String, String>();
                // adding each child node to HashMap key => value
                map.put(TAG_DEPID, depid);
                map.put(TAG_DEPARTMENT, department);
                // adding HashList to ArrayList
                catchdepartmentsList.add(map);
            }
        }
    }
}
}

```



```

public static final String TAG_NAME = "name";
public static final String TAG_MACADDRESS = "macaddress";
private static final String TAG_DEPID = "depid"; // weg?
String depid;
String phoneid;
// products JSONArray
JSONArray catchphones = null;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.all_phones);
    // getting product details from intent
    Intent j = getIntent();
    // getting department from intent
    depid = j.getStringExtra(TAG_DEPID);
    // Hashmap for ListView
    catchphonesList = new ArrayList<HashMap<String, String>>();
    // Loading products in Background Thread
    new LoadAllPhones().execute();
    // Get listview
    ListView lv = getListView();
    // on selecting single product
    // launching Edit Product Screen
    lv.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view,
            int position, long id) {
            // getting values from selected ListItem
            String phoneid = ((TextView)
view.findViewById(R.id.phoneid)).getText().toString();
            String name = ((TextView)
view.findViewById(R.id.name)).getText().toString();
            String macaddress = ((TextView)
view.findViewById(R.id.mac)).getText().toString();
            // reference naar list_item
            //create bundle
            Bundle extras = new Bundle();
            // fill bundle
            extras.putString("TAG_PHONEID",phoneid);
            extras.putString("TAG_NAME",name);
            extras.putString("TAG_MACADDRESS",macaddress);
            // Starting new intent Select Phone Activity
            Intent in = new Intent(getApplicationContext(),
                SelectPhoneActivity.class);
            // sending bundle to next activity
            in.putExtras(extras);
            // starting new activity and expecting some response back
            startActivityForResult(in, 100);
        }
    });
}

```

```

    });
}
// Response from Edit Product Activity
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    // if result code 100
    if (resultCode == 100) {
        Intent intent = getIntent();
        finish();
        startActivity(intent);
    }
}
/**
 * Background Async Task to Load all product by making HTTP Request
 */
class LoadAllPhones extends AsyncTask<String, String, String> {
    /**
     * Before starting background thread Show Progress Dialog
     */
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressDialog = new ProgressDialog(AllPhonesActivity.this);
        progressDialog.setMessage("Loading phones. Please wait...");
        progressDialog.setIndeterminate(false);
        progressDialog.setCancelable(false);
        progressDialog.show();
    }
    /**
     * getting All phones from url
     */
    protected String doInBackground(String... args) {
        // Building Parameters
        List<NameValuePair> params = new ArrayList<NameValuePair>();
        params.add(new BasicNameValuePair("depid", depid));
        // getting JSON string from URL
        JSONObject json = jParser.makeHttpRequest(url_all_phones, "GET", params);
        // Check your log cat for JSON reponse
        Log.d("All Phones: ", json.toString());
        try {
            // Checking for SUCCESS TAG
            int success = json.getInt(TAG_SUCCESS);
            if (success == 1) {
                // products found
                // Getting Array of Products
                catchphones = json.getJSONArray(TAG_CATCHPHONES);
                // looping through All Products
                for (int i = 0; i < catchphones.length(); i++) {

```



```

// Creating JSON Parser object
JSONParser jParser = new JSONParser();
ArrayList<HashMap<String, String>> catchphonedetailsList;
// JSON Node names
String phoneid;
// products JSONArray
JSONArray catchphonedetails = null;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.select_phone);
    // getting phone details from intent
    // getting bundle from intent
    Bundle extras = getIntent().getExtras();
    // get string messages
    String name = extras.getString("TAG_NAME");
    String macaddress = extras.getString("TAG_MACADDRESS");
    // find own wifi macaddress
    WifiManager manager = (WifiManager)
getSystemService(Context.WIFI_SERVICE);
    WifiInfo info = manager.getConnectionInfo();
    String address = info.getMacAddress();
    //select textviews
    TextView txtPhoneName = (TextView)findViewById(R.id.txtPhoneName);
    TextView txtPhoneMac = (TextView)findViewById(R.id.txtPhoneMac);
    // set text and content
    txtPhoneName.setText(name);
    txtPhoneMac.setText(macaddress);
    TextView txtOwnMac = (TextView)findViewById(R.id.txtOwnMac);
    txtOwnMac.setText(address);
    // Button
    Button btnGetRoute = (Button) findViewById(R.id.btnGetRoute);
    // view products click event
    btnGetRoute.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            // Launching GetRouteActivity
            Intent i = new Intent(getApplicationContext(), GetRouteActivity.class);
            startActivity(i);
        }
    });
}
// Response from Edit Product Activity
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    // if result code 100
    if (resultCode == 100) {
        Intent intent = getIntent();

```

```

        finish();
        startActivity(intent);
    }
}

```

GetRouteActivity.java

```

package com.example.derotterdamappsql;
public class GetRouteActivity extends Activity {
    Button btnMapView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.get_route);
        // Button
        Button btnMapView;
        btnMapView = (Button) findViewById(R.id.btnMapView);
        // view map view click event
        btnMapView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Launching All products Activity
                Intent i = new Intent(getApplicationContext(), MapViewActivity.class);
                startActivity(i);
            }
        });
    }
}
MapViewActivity.java

```

```

package com.synthesis.deRotterdamApp;
public class MapViewActivity extends UnityPlayerActivity {
    public static Context mContext;
    @Override
    protected void onCreate(Bundle bundle)
    {
        super.onCreate(bundle);
        mContext = this;
    }
}

```

routeRenderer.js

```
#pragma strict
```

```
var nodeInfo : TextAsset;
```

```
function Start () {
```

```
    var routeArray = [1408,1409,1404,1405];
```

```
    var routeVectorArray = new Array();
```

```
    var lines = nodeInfo.text.Split("\n"[0]);
```

```
    var nodeInfoLength : int = lines.Length;
```

```
    // MOVE CORRECT GEOMETRY INTO VIEW
```

```
    var finalNode : int = routeArray[routeArray.Length-1];
```

```
    var finalFloor : int = (finalNode/100);
```

```
    for (var geomIndex : int = 14; geomIndex < 17; geomIndex++)
```

```
        { Debug.Log(geomIndex);
```

```
          if (finalFloor != geomIndex)
```

```
              { var tempGeom : GameObject = GameObject.Find("RdamFloor" +  
geomIndex); tempGeom.SetActive (false); }
```

```
        }
```

```
    // READ NODEINFO & FILL ROUTEVECTORARRAY
```

```
    for(var i : int = 0; i < routeArray.length; i++)
```

```
        {
```

```
            var tempNode = routeArray[i];
```

```
            for(var j : int = 1; j < nodeInfoLength; j++)
```

```
                {
```

```
                    var nodeParts = lines[j].Split(";"[0]);
```

```
                    var nodeID = int.Parse(nodeParts[0]);
```

```
                    if (nodeID == tempNode)
```

```
                        { var nodeX = float.Parse(nodeParts[1]);
```

```
                          var nodeY = float.Parse(nodeParts[2]);
```

```
                          var nodeZ = float.Parse(nodeParts[3]);
```

```
                          var nodeVector : Vector3 = Vector3(nodeX,nodeY,nodeZ);
```

```
                          routeVectorArray.Push(nodeVector);
```

```
                        }
```

```
                }
```

```
            }
```

```
    // CREATE GAMEOBJECT, ADD LINERENDERER AND SET POSITIONS /  
COLORS
```

```
    var lineObjectArray = new Array();
```

```
    for(var index : int = 0; index < routeVectorArray.length-1; index++)
```

```
        {
```

```
            Debug.Log("pair" + index.ToString());
```

```
            var pairStart : Vector3 = routeVectorArray[index];
```

```
            Debug.Log(pairStart);
```

```
            var pairEnd : Vector3 = routeVectorArray[index+1];
```

```
            Debug.Log(pairEnd);
```

```
            var tempObject : GameObject = new GameObject();
```

```
            var renderer : LineRenderer = tempObject.AddComponent(LineRenderer);
```

```
            renderer.SetVertexCount(2);
```



```

        renderer.SetWidth(0.5f,0.5f);
        renderer.SetPosition(0, pairStart);
        renderer.SetPosition(1, pairEnd);
        var blueMat : Material = renderer.material;
        var mat : Material = Resources.Load("blueLine", typeof(Material) ) as
Material;
        renderer.material = mat;
        blueMat.shader = Shader.Find("Custom");
    }

    //CREATE SPHERES
    //startnode
    var startNode : GameObject = GameObject.CreatePrimitive(PrimitiveType.Sphere);
    var startVector : Vector3 = routeVectorArray[0];
    startNode.renderer.material.color = Color(0.2,0.6,0.8,0);
    startNode.transform.position = startVector;
    //middlenodes
    for(var index1 : int = 1; index1 < routeVectorArray.length-1; index1++)
    {
        var middleNode : GameObject =
GameObject.CreatePrimitive(PrimitiveType.Sphere);
        var middleVector : Vector3 = routeVectorArray[index1];
        middleNode.renderer.material.color = Color(1,1,1,0);
        middleNode.transform.position = middleVector;
    }
    // end node
    var endNode : GameObject = GameObject.CreatePrimitive(PrimitiveType.Sphere);
    var endVector : Vector3 = routeVectorArray[routeArray.Length-1];
    endNode.renderer.material.color = Color(0.2,0.8,0.0,0);
    endNode.transform.position = endVector;

    }
    function Update () {
    }

```

```

rightButtonScript.js    Similar code is created for all zoom and pan buttons
function OnMouseDown(){
    print("down");
}
function OnMouseUp(){
    print("up");
    var currentX : float = Camera.main.transform.position.x;
    var currentY :float = Camera.main.transform.position.y;
    var currentZ :float = Camera.main.transform.position.z;
    var stepSize : float = 30/currentY;
    Camera.main.transform.position = Vector3(currentX+stepSize, currentY, currentZ);
}

```

## Get\_all\_departments.php

<?php

```
/*
 * Following code will list all the departments
 */

// array for JSON response
$response = array();

// include db connect class
require_once __DIR__ . '/db_connect.php';

// connecting to db
$db = new DB_CONNECT();

// get all departments from catchdepartments table
$result = mysql_query("SELECT *FROM catchdepartments") or die(mysql_error());

// check for empty result
if (mysql_num_rows($result) > 0) {
    // looping through all results
    // departments node
    $response["catchdepartments"] = array();

    while ($row = mysql_fetch_array($result)) {
        // temp user array
        $catchdepartment = array();
        $catchdepartment["depid"] = $row["depid"];
        $catchdepartment["department"] = $row["department"];
        $catchdepartment["created_at"] = $row["created_at"];
        $catchdepartment["updated_at"] = $row["updated_at"];
        // push single department into final response array
        array_push($response["catchdepartments"], $catchdepartment);
    }
    // success
    $response["success"] = 1;

    // echoing JSON response
    echo json_encode($response);
} else {
    // no departments found
    $response["success"] = 0;
    $response["message"] = "No departments found";

    // echo no users JSON
    echo json_encode($response);
}
```

?>

Get\_department\_details.php

<?php

```
//Following code will list all the phones of a single department
// array for JSON response
$response = array();
// include db connect class
require_once __DIR__ . '/db_connect.php';
// connecting to db
$db = new DB_CONNECT();
// check for post data line 80 in alldepartmentsactivity
if (isset($_GET["depid"])) {
    $depid = $_GET['depid'];
    // get a department from departments table
    $result = mysql_query("SELECT * FROM catchphones WHERE depid = $depid");
    // $result - mysql_query("SELECT * FROM catchphones WHERE depid
    if (!empty($result)) {
        // check for empty result
        if (mysql_num_rows($result) > 0) {
            // looping through all results phones node
            $response["catchphones"] = array();
            while ($row = mysql_fetch_array($result)) {
                $catchphone = array();
                $catchphone["phoneid"] = $row["phoneid"];
                $catchphone["name"] = $row["name"];
                $catchphone["macaddress"] = $row["macaddress"];
                $catchphone["department"] = $row["department"];
                $catchphone["depid"] = $row["depid"];
                $catchphone["created_at"] = $row["created_at"];
                $catchphone["updated_at"] = $row["updated_at"];
                // push single phone into final response array
                array_push($response["catchphones"], $catchphone);
            } // success
            $response["success"] = 1;
            echo json_encode($response);
        }
    } else {
        // no phone found
        $response["success"] = 0;
        $response["message"] = "No phone found";
        // echo no users JSON
        echo json_encode($response);
    }
} else {
    // required field is missing
    $response["success"] = 0;
    $response["message"] = "Required field(s) is missing";
}
```

```

    // echoing JSON response
    echo json_encode($response);
}
?>
db_connect.php

```

```

<?php
/**
 * A class file to connect to database
 */
class DB_CONNECT {
    // constructor
    function __construct() {
        // connecting to database
        $this->connect();
    }
    // destructor
    function __destruct() {
        // closing db connection
        $this->close();
    }
}
/**
 * Function to connect with database
 */
function connect() {
    // import database connection variables
    require_once __DIR__ . '/db_config.php';
    // Connecting to mysql database
    $con = @mysql_connect(DB_SERVER, DB_USER, DB_PASSWORD) or
die(mysql_error());
    // $con = mysqli_connect(DB_SERVER, DB_USER, DB_PASSWORD) or
die(mysql_error());
    // Seleccioning database
    $db = mysql_select_db(DB_DATABASE) or die(mysql_error()) or
die(mysql_error());
    // returning connection cursor
    return $con;
}
/**
 * Function to close db connection
 */
function close() {
    mysql_close();
}
}
?>

```

Db\_config.php

```
<?php
/*
 * All database connection variables
 */
define('DB_USER', "XXXXX"); // db user
define('DB_PASSWORD', , "XXXXX"); // db password (mention your db password here)
define('DB_DATABASE', , "XXXXX"); // database name
define('DB_SERVER', "XXXXX"); // db server
?>
```