

## Indoor Localisation and Navigation

Thomas Carroll

Kane Easby

Kevan Jordan

Tomáš Martínek

Samuel Palabiyik

Matthew Pawson

Usama Usman

Submitted in accordance with the requirements for the degree of

MEng Computer Science

2021/2022

30 credits

The candidates confirm that the following have been submitted.

Items	Format	Recipient (Date)
Report	PDF	Minerva (April 28, 2022)
Source code	URL	Assessor (April 28, 2022)
Video demo	URL	Assessor (April 28, 2022)

The candidates confirm that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

We understand that failure to attribute material which is obtained from another source may be considered as plagiarism.



Thomas Carroll



Kane Easby



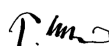
Kevan Jordan



Samuel Palabiyik



Matthew Pawson



Tomáš Martínek



Usama Usman

## Summary

Indoor navigation as a concept has existed for many years. However, there are very few examples being used in human-occupied environments. Similarly to standard outdoor navigation, it provides users with their location and a route to their required destination. It also introduces new challenges to overcome.

One particular challenge is how to actually locate a user within a building. Most current systems are built for outdoor navigation and use GPS, which is not suitable for indoor navigation due to interference caused by the surrounding environment.

Another challenge is mapping itself, buildings introduce a third dimension. Whilst traditional navigational maps account for a 2D space, they are not required to take into account the multiple elevations a building contains. Currently there are no consistent methods of implementing this. Not only will the mapping have to take into account this third dimension, localisation and navigation will also need to overcome this problem such that a system can tell users which floor of the building they are on.

This project will focus on indoor navigation, mapping and localisation. Our aim is to develop a system of mapping, storing, and localising a user using the existing systems already available. For each of these systems we hope to develop concepts that will provide users with a better navigation system than traditional signs and static maps. This project will also be constrained by the limitation of our budget. Our plan is to implement this system without the need for additional hardware.

The bulk of the project can be split into three distinct groups: mapping, server and localisation. Throughout this report we will identify aspects of these sections, methods we used to implement the systems, and how we overcame any potential challenges.

## Acknowledgements

We would like to thank Dr. Evangelos Pournaras for his guidance and supervision, and Professor Raymond Kwan for his advice and help organising this project.

We would also like to thank:

- Dr Sam Wilson, for providing floor plans and Wi-Fi router maps
- OpenStreetMap Project, for lots of very important information and thoughts about mapping
- All experiment participants



# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Problem . . . . .	2
1.2	Project Aim . . . . .	3
1.3	Objectives . . . . .	3
1.4	Deliverables . . . . .	4
1.5	Ethics . . . . .	4
<b>2</b>	<b>Planning</b>	<b>5</b>
2.1	Methodology . . . . .	5
2.1.1	Issues with Project Management . . . . .	6
2.2	Plan . . . . .	6
2.3	Version Control . . . . .	7
2.3.1	Branch Strategy . . . . .	7
2.3.2	Continuous Integration . . . . .	7
<b>3</b>	<b>Mapping</b>	<b>8</b>
3.1	Initial Ideas . . . . .	8
3.1.1	Existing Solutions . . . . .	8
3.1.2	OpenStreetMaps . . . . .	11
3.2	Making the Map . . . . .	11
3.2.1	Levels and Stairs . . . . .	12
3.2.2	Tags . . . . .	13
3.3	File Formats . . . . .	13
3.4	Further Work . . . . .	14
<b>4</b>	<b>Localisation</b>	<b>15</b>
4.1	Existing Solutions . . . . .	16
4.2	Possible Solutions . . . . .	17
4.3	Proposed Solution . . . . .	18
4.4	Implementation . . . . .	19
4.4.1	Recording Router Information . . . . .	19
4.4.2	Trilateration . . . . .	20
4.5	Tuning the Parameters . . . . .	22
4.5.1	Data Collection . . . . .	23
4.5.2	Search Method . . . . .	24
4.5.3	Implementation . . . . .	25
4.5.4	Parallel Processing . . . . .	28
4.5.5	Outcomes . . . . .	31
4.6	Future Work . . . . .	33

<b>5</b>	<b>Server</b>	<b>34</b>
5.1	Architecture . . . . .	34
5.1.1	Parser . . . . .	35
5.1.2	Database . . . . .	35
5.1.3	API . . . . .	36
5.1.4	Path-finding . . . . .	37
5.2	Build & Deploy . . . . .	37
5.3	Further Work . . . . .	39
<b>6</b>	<b>App</b>	<b>41</b>
6.1	Implementation . . . . .	41
6.2	Design . . . . .	41
6.2.1	Technical . . . . .	41
6.2.2	UI / UX . . . . .	43
6.3	Design Iterations . . . . .	46
6.4	Further Work . . . . .	48
<b>7</b>	<b>Evaluation</b>	<b>49</b>
7.1	Experiment . . . . .	49
7.1.1	Sampling . . . . .	49
7.1.2	Metrics . . . . .	49
7.1.3	Search Locations . . . . .	50
7.1.4	Survey . . . . .	50
7.1.5	Baseline Experiment . . . . .	50
7.1.6	App Experiment . . . . .	51
7.2	Results . . . . .	51
7.2.1	Baseline Survey . . . . .	51
7.2.2	App Survey . . . . .	53
7.2.3	Comparison of Results . . . . .	55
7.2.4	Survey Comparison . . . . .	58
7.3	Revised Procedures . . . . .	59
<b>8</b>	<b>Conclusion</b>	<b>60</b>
8.1	Mapping . . . . .	60
8.2	Localisation . . . . .	60
8.3	Implementation . . . . .	60
8.4	Teamwork . . . . .	61
	<b>References</b>	<b>62</b>
	<b>Appendices</b>	<b>65</b>
<b>A</b>	<b>Source Code</b>	<b>66</b>
A.1	Project Repository . . . . .	66
A.2	Map Repository . . . . .	66

A.3	Video demo . . . . .	66
A.4	Survey . . . . .	66
A.5	Consent Forms . . . . .	66
A.6	Experiment Data & Survey Results . . . . .	66
<b>B</b>	<b>List of Libraries / Software Used</b>	<b>67</b>
B.1	Server . . . . .	67
B.1.1	Software . . . . .	67
B.1.2	Python libraries . . . . .	67
B.2	Client . . . . .	67
<b>C</b>	<b>Parameter Tuning Graphs</b>	<b>69</b>
<b>D</b>	<b>Parameter Tuning Tables</b>	<b>71</b>
<b>E</b>	<b>Listings</b>	<b>72</b>
<b>F</b>	<b>Evaluation</b>	<b>74</b>
F.1	Consent Form . . . . .	74
F.2	Information Sheet . . . . .	75
F.3	Invigilator Instructions . . . . .	77
F.4	Participant Instructions — Baseline . . . . .	78
F.5	Participant instructions — App . . . . .	79
F.6	Survey Questions . . . . .	80

# Chapter 1

## Introduction

### 1.1 Problem

Finding your way inside large buildings, such as offices and university campuses, is hard. We have found that the newly constructed Sir William Henry Bragg building is no exception. It can be a struggle to find a certain room using signs or static maps, as they can be misleading or confusing. For example, Figure 1.1 shows signs that are both incorrect and inconsistent with each other.

Signs and printed maps require constant updates as the building changes over time. Even temporary changes, such as blocking corridors for maintenance or restricting room access out-of-hours, need to be communicated in a clear and timely manner. Doing this requires a person to physically access the building and change the signage.

These systems are also not very accessible. Visitors with disabilities, or people whose first language is not English may not be able to read signs easily, or ask people in the building for help.

We can see that a digital system would be able to help in these scenarios, even if we do not directly implement these accessibility features. Creating a framework that is open for extension means a reduction in time and effort as there is only one central repository for building data. Some parts may even be automated (e.g. showing room bookings, or updating office allocations dynamically).

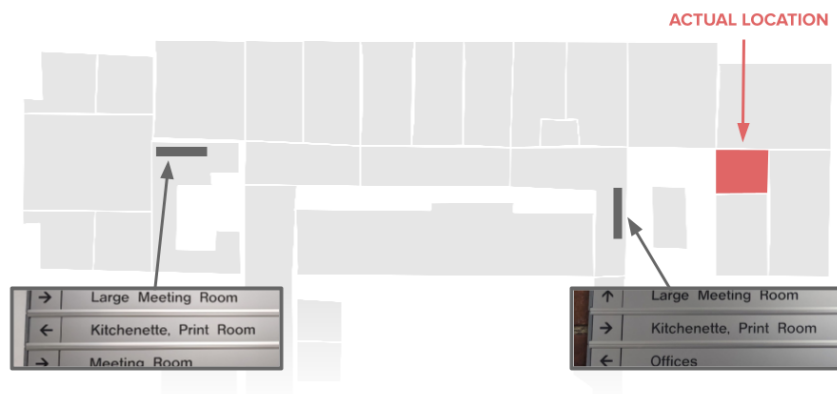


Figure 1.1: Two signs that are completely wrong.

## 1.2 Project Aim

This project aims to create a solution that provides users with a method for navigating buildings. The proposed solution will involve creating a scheme for mapping buildings and use Wi-Fi-based localisation. These maps will be served using a web service that a mobile app will connect to. The user should be able to enter their destination and be shown a path from their current location they can follow.

To be successful, this app should provide users with a method that does not only meter level accurate but in conjunction is a better method of finding their way around the building. We will evaluate this with user testing by running an experiment testing one group without the app, and one without.

## 1.3 Objectives

For this project, our main objectives are to:

- Produce a scheme for indoor mapping and navigation
- Define and implement an API for map retrieval
- Implement indoor localisation using wireless access points
- Create an application for indoor navigation using our localisation method
- Evaluate our application through user testing

A method for mapping commercial buildings will be produced. This method should be suitable for navigation primarily; precise details should not be recorded, but enough information should be present to allow users to contextualise their location within the map. Further, this method should allow for a ‘bottom-up’ creation of maps by users that might not have access to the architectural plans. This would allow users to produce their own maps similar to the Open Street Maps project, allowing maps of buildings to be crowd-sourced.

A method for localisation within a building will be produced. This should not use any additional hardware; instead, it will extend upon existing functionality within mobile devices. This method will use Wi-Fi access points and trilateration based on signal strength to locate the user relative to a number of access points. Not using extra hardware will keep the cost of the project to zero, whilst allowing the scheme to be used in any buildings that have multiple access points.

Server software that takes the map files and stores them in a database, for use by client implementations, should be created. This server will be agnostic to the specifics of the data so that client implementations can be created when requirements for the mapping data are changed. This means that the server should only use the data that is absolutely necessary, to allow client implementations to have their own rules around how the maps are displayed.

The server should also implement a basic routing service that clients can use to find a path through the building.

Finally, a reference client mobile app should be produced that allows users to view the map and navigate through it. This client app should implement the localisation method and use the features of the mapping method to allow for this navigation.

## 1.4 Deliverables

The deliverables for this project are:

- This report, containing systems for mapping and localisation
- Source code for the server and app, implementing these systems
- Video demo of the app

## 1.5 Ethics

Creating tools that empower any person to create and distribute maps of buildings, potentially against the owners wishes, is a potential ethical issue. We do not believe that this is a large enough problem to weigh against the project as we are using techniques that could have already have been conceived and executed. For example, there is nothing to stop someone making a sketch of a building already. If there are legal issues with mapping certain buildings this is another issue to deal with at a larger scale; as it stands our project only contains maps of one building, of which the floor plans are already freely available online.

Using a Wi-Fi based localisation method comes with no ethical issues. A potential concern in this area may be if the app connected to networks without authorisation. Our method only uses the Wi-Fi beacon frames; these are broadcast ‘advertising’ messages that announce the access point’s information. These facilitate a device being able to detect networks, which does not require a connection or authentication. Hence, we would not connect to networks that we are not authorised to do so.

When running user experiments it is important that General Data Protection Regulation (GDPR) is acknowledged [21]. It should be ensured that all participants in the experiments both know how their data will be used, and have the ability to withdraw at any time. We should also ensure we are gathering informed consent for their data to be used in this study.

# Chapter 2

## Planning

### 2.1 Methodology

The project adopted an Agile methodology [5], with Rapid Application Development (RAD) as an agile project management strategy and SCRUM technique.

#### SCRUM

A classic Agile methodology centred around sprints that requires a SCRUM master — a person who represents the stakeholders and is not involved in development [28].

#### RAD

A project management strategy that is specifically designed for creating components quickly after an initial planning phase [19]. Unlike SCRUM, the RAD strategy allows for changes in strategy during sprints rather than at the end. In the RAD methodology, there are four phases: a planning phase, a user design phase, a series of sprints focused on creating components, and a cut-over phase where the product is released.

Agile in general is more flexible than traditional development methodologies (e.g. Waterfall) due to its ability to account for errors produced within the planning phase. This added flexibility is especially important for research projects, where unexpected obstacles are often encountered. Agile methodologies are good in places where the final requirements are not clear from the outset, and may change over time. However, it came with new challenges as the success of the project relied on self-organising skills and commitment of individuals.

Throughout the project we had group meetings every week and a supervisor meeting every two weeks. Regular meetings ensured that the group was familiar with the state of the project, and that progress was made consistently. New functionalities were demonstrated on a regular basis in both group and supervisor meetings.

During development of the app we used the RAD framework:

- The app features were updated rapidly in separate branches to prevent merge conflicts
- The integrity of every commit was checked through the use of DevOps and CI, in order to prevent code changes from breaking the app
- The whole implementation was tracked with branches being merged when complete/no longer required

### 2.1.1 Issues with Project Management

We began the project with SCRUM, but moved on from it later on. Our reasoning for this was that we could not host daily stand-ups due to schedule conflicts, and we could not separate the SCRUM master from the development as that would reduce the number of developers from our app increasing the workload on the other group members

We then transitioned to using Discord — a popular instant messaging platform [11]. We set up a Discord group for day-to-day communication and sharing resources, however, Discord did not help with prioritisation and important information could be missed due to the overwhelming number of messages. To mitigate this, we used sub-channels, pinning and member tagging.

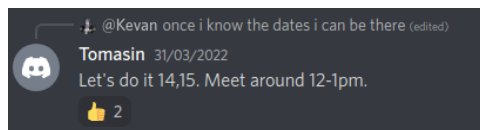


Figure 2.1: Organising a meeting using Discord.

The Mythical Man-Month says “The added burden of communication is made up of two parts, training and intercommunication.” [6] This goes beyond verbal communication, and includes all aspects of software engineering. In our case, the ‘training’ part involved setting up the development environment, learning a new framework and standardising coding style. ‘Intercommunication’ includes all forms of communication, as well as branching and version control of the whole project.

## 2.2 Plan

We created a Gantt chart to show the time allocations for each part of the project (see Figure 2.2). Most components were developed simultaneously by groups of 2-3 people and reviewed regularly. This setup allowed us to iterate quickly over the tight 6-month schedule.

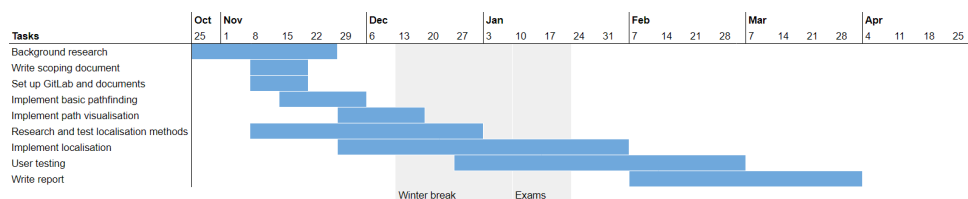


Figure 2.2: Gantt chart showing the initial plan of the project



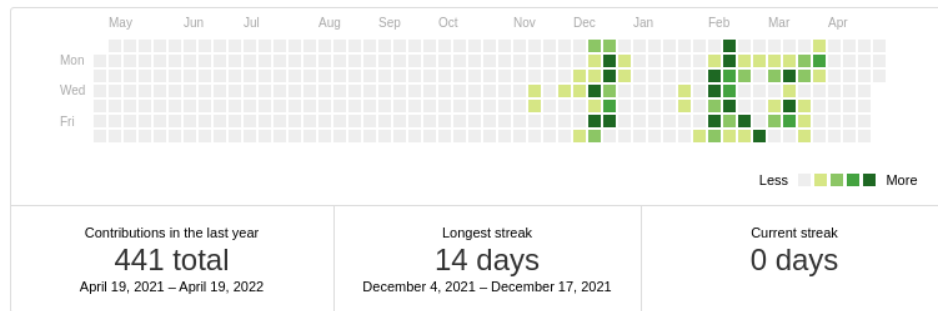


Figure 2.3: Git stats for the project

## 2.3 Version Control

Version control is necessary for collaborative team projects to ensure that members are up-to-date with software implementation and able to revert unwanted changes. GitLab was used throughout the project to keep track of changes and updates.

### 2.3.1 Branch Strategy

All development was carried out in feature branches, and merged to `dev` when completed. The `dev` branch was used for testing before merging to `main`. This helped to ensure that the code in the `main` branch was always stable and could be deployed at-will.

### 2.3.2 Continuous Integration

We added automated linting and testing through GitLab’s Continuous Integration (CI) feature to ensure high-quality code. Code would only be merged if it passed all the pipeline checks. This ensured that a commit did not cause a regression and that code style remains consistent. Chapter 5 covers this in more detail. We set up a pre-commit hook to format code before pushing to GitLab, which helped to reduce pipeline failures.

Listing 2.1: Pre-commit hook

```
#!/bin/sh

cd app
npm run lint -- --fix

cd ..
black server
```

# Chapter 3

## Mapping

### 3.1 Initial Ideas

To use path finding algorithms a graph representation of a building needs to be developed; there are multiple methods to create such a representation of a building. For example, a naive approach would be to put a node in the centre of every room and put an edge between nodes if the rooms have a door in between. This has a few issues: distances between nodes are not represented, and hallways can not be represented easily.

#### 3.1.1 Existing Solutions

Maps are an ancient technology, there are thousands of years of existing solutions to consider. The art of cartography (and it is an art) is also deeply entwined with scientific progress. All cartography creates an abstraction away from the real ‘ground-truth’ of the world; in Figure 3.1, we see a map that is purposefully more narrative and spiritual. In the same way, our map needs to be accurate enough to allow for navigation, but does not need to represent every millimeter of the building.



Figure 3.1: Hereford Mappa Mundi

It is useful to keep in mind that our map is a combination of geometric and semantic data. The Bragg building’s floor plans only contain the geometric information. We have to assign semantic meaning (e.g. room names, their functions) to each of the geometries.

Richard Atterer proposes another interesting solution for creating maps without access to the floor plans of a building. He attached two gaming mice to the wheels of a suitcase to act as an ‘augmented trundle wheel’ that can be used to accurately map the paths that one might take around a building [4].

He also mentions recording Wi-Fi signal strength at multiple points in the building. This is more of a bottom-up approach than the top-down approach that we have taken. Since we have the floor plans and the positions of all the Wi-Fi routers, we do not need to map the building ourselves.

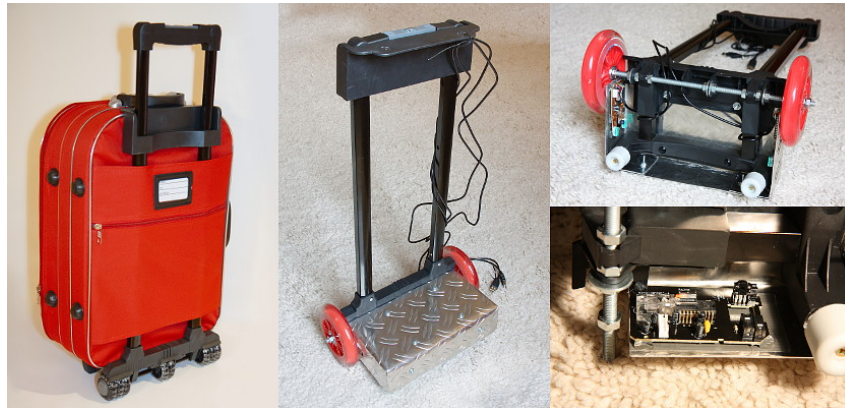


Figure 3.2: Richard Atterer’s ‘Position Mouse’ [4]

This does show however that the art of indoor cartography has not been fully explored. Since this project in 2010, there has not been much more research towards a similar bottom-up method of creating maps.

That is not to say that the art of indoor cartography is not a deep and rich field, but the idea of a navigation-forward approach perhaps is not as much as the approach taken by architects or interior designers. These cartographers are much more focused on the experience and emotion of being in a building, while we are more interested in how people move through and interact with the space. A review paper published in 2019 states that “[w]hile CAD and BIM provide greater support for symbolization (Chen & Clarke, 2017; Petrie, 2016), their niche focus on building construction ends up working against sound cartographic principles due to excessive detail, limited geometric and semantic flexibility.” [9]

This shows us that maps of buildings serve many different purposes, where a retailer may want to create a map with the purpose of promoting and selling goods, a factory owner may want to create a ‘digital twin’ of their factory for simulation and training. Hence, we should consider exactly what the creation of our map is seeking to achieve and what it should represent to the user.

It is also important to consider the mental models that people make of the building. Do they consider it as a collection of 3D spaces first and foremost (this is what the architect may like to think) or is it acceptable to represent the building as we might a road or park on a paper map?

The same review paper [9] compares a variety of companies’ approaches to indoor map representations, with all but one (‘WRLD’) approaching the problem with 2D maps. For

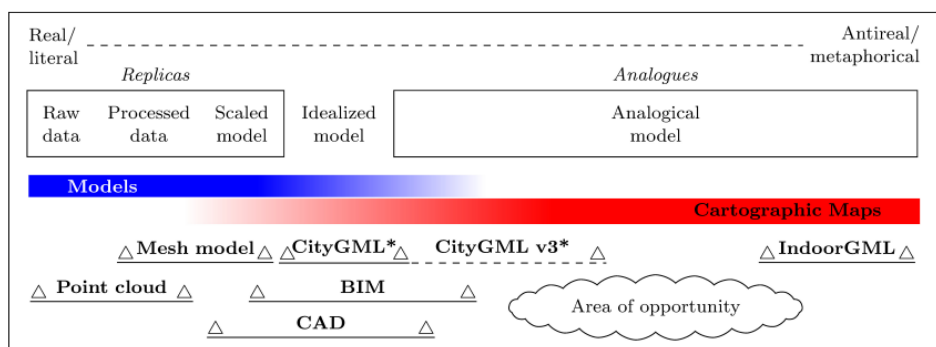


Figure 3.3: Aisle411 in-store mapping [8]

example, Aisle 411 has created an add-on screen for shopping trolleys that helps users to find products in retail stores [Figure 3.3].

Since these applications are for commercial environments, they exist in a much more top-down methodology; there is very little scope for more ‘clandestine’ mapping, à la Atterer [4]. These maps can be created by anyone, similarly to OpenStreetMap, where everybody has the power to collaborate in the act of cartography.

For these maps, a popular data format is IndoorGML, a standard developed by the Open Geospatial Consortium (OGC). This is a fairly low-level format, but includes concepts like semantic and geometric representation. The space is broken down into multiple cells, which represent ‘the smallest organisational or structural unit of indoor space’ [22]. However, this is an incredibly complex standard and is infeasible for this project. It focuses more on the topological relation of these cells, rather than the geometric relation we want for navigation.



\* Indoor LOD(s) only

Figure 3.4: Current approaches to representing indoor spaces in the context of mapping and modelling (Reproduced from [9])

Another option is OpenStreetMap’s proposed indoor mapping tags [24]. This would enable us to use our maps in collaboration with the OSM project. This gives a good compromise between measured data (point clouds) and human-centric data (paper maps), placing this in the ‘area of opportunity’ above (Figure 3.4).

### 3.1.2 OpenStreetMaps

We used QGIS, a free and open source software suite, to create OSM-compatible maps known as ‘shapefiles’<sup>1</sup> that define geospatial data. These files contain points, lines and polygons, all of which have associated metadata.

OSM has its own representation of paths that it uses for roads, footpaths, etc. that it calls ‘Ways’ [23]. These represent linear traversable paths, an abstraction away from roads, paths and highways. In OSM, these ways can have attributes that change how they are displayed, and how they affect navigation.

We must also recognise that we are choosing the ‘room’ or polygon as the basic building block of a building. Whilst this makes sense for the Henry Bragg building, does it fully represent every building that one might want to map? For instance, it may not be the perfect model for a sports stadium, which has few rooms but many distinct locations that one might want to find a path to (e.g. a specific seat in the stands).

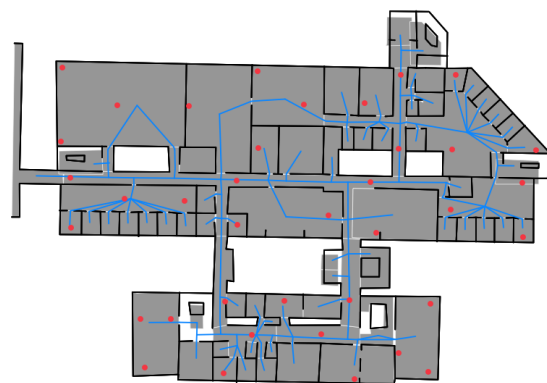
Whilst ‘ways’ are generally suitable for a building, we need to consider that a building is not in fact a series of connected linear paths, but is instead multiple contiguous spaces. For example, this system may work well for commercial buildings, but does not make much sense for smaller buildings. Therefore, we have chosen to implement this standard because it works well with our choice of buildings (buildings on a university campus) and the larger OSM project.

## 3.2 Making the Map

To relate the floor plans we were given to the geographical coordinates we used the QGIS ‘georeferencer’ plug-in. We took measurements at known points around the outside of the building using GPS and referenced these to the floor plans we were given. The plug-in then re-projects the reference image to the correct position and rotation of the world map.



(a) Geo-referenced floor plan for Bragg Building  
Floor 2



(b) Floor plan after features digitised

Figure 3.5: Floor plans of the Bragg building before and after digitising

<sup>1</sup>In fact, OSM has its own special XML-like format it uses, but they are roughly analogous.

Then, all the rooms and walls were traced digitally from the reference images and labelled with their metadata. These features are then organised into separate layers representing floors of the building. Paths between rooms are represented as ‘ways’, and are annotated with their type (hallway, stairs, etc). To represent intersections, we use the ‘split on line’ option in QGIS, which ensures that intersecting lines share a common point.

### 3.2.1 Levels and Stairs

Buildings often have more than one floor. To represent floors in our map, all features have a ‘level’ attribute containing the floor that they reside on. Hence, all features can be stored in one file and still be filtered by floor. This is the suggested solution from OSM’s ‘Indoor Mapping’ wiki page [24].



Figure 3.6: Stairs and Lifts in the Bragg Building

For stairs and lifts, we used a single attribute that specifies the range of floors (i.e. ‘2;5’ connects floors 2 through 5) they are connected to. Since these are the only features that span across floors, we can use them directly in existing path finding algorithms. This is a slight deviation from the recommended approach, which uses multiple ‘level’ attributes. Staircases and lifts are tagged differently so path finding can be handled differently in each case.

Some staircases look strange in Figure 3.6 because we trace the shape of the staircase at the bottom floor. In the case of *Stair 1 GR.A07*, the shape on the ground floor is different than on the other floors. To fix this issue, we would need to specify per-floor geometries of the staircases, however, this is not possible with QGIS.



### 3.2.2 Tags

We defined optional tags for each of the elements in the map, these were in line with OSM’s existing tagging system, and many elements (for example Wi-Fi routers) already have well-defined schemas. The server exists to be as agnostic to the map data as possible, allowing the client to implement the way it handles tags. These tags enable additional functionality as explained in Chapter 5.

#### Polygon Tags

Attribute Name	Type	Example	Note
room-no	String	“GR.A07”	
room-name	String	“Wormery”	
indoor	String	“room”	One of: corridor, area, room <sup>2</sup>
stairs	String	“yes”	Stairs is set only on stairs, OSM-compatible
highway	String	“elevator”	Only used for elevators, OSM-compatible
level	String	1;3	Represents multiple floors (floors 1 to 3)

#### Linestring Tags

Attribute Name	Type	Example	Note
indoor	String	“wall”	Type of line, one of: wall, way
level	Float	0	

#### Point Tags

Attribute Name	Type	Example	Note
amenity	String	“water fountain”	PoI tag, OSM-compatible
mac_addres	String	“ac:3a:67:08:25:a0”	
SSID	String	“wap70254”	
internet	String	“yes”	Indicates whether this is a WAP
door	String	“no”	Doorway: can be ‘yes’, ‘no’ <sup>3</sup>
indoor	String	“door”	Only used for doors, OSM-compatible
level	String	0	

## 3.3 File Formats

QGIS produces files in the shapefile format (`.shp`), which has one file for each layer in the project. To convert these to other formats, we can use a utility called `ogr2ogr`, which is part of the `gdal` library.

The `.shp` format is an old format and has certain limits that we did not realise in early stages. One of these is a 10-character limit on the name of the field, which means that we had to truncate `mac_address` to `mac_addres`. On reflection, we should have used a more modern format, but it was too late to change our approach.

<sup>2</sup>‘Type’ of room , this is for path finding, and was added to try and facilitate path generation

<sup>3</sup>‘no’ confusingly means just the threshold, can be more specific e.g. ‘revolving’

Since the `.shp` files have a binary format and hard to parse, we chose to use an intermediate data format. We chose GeoJSON as it is easy to parse and preserves all the features. We automated this by setting up a GitLab CI pipeline that takes these `.shp` files and converts them to GeoJSON. We also created a script that automatically downloaded the most up-to-date GeoJSON artifacts. We used a separate Git repository, `example_maps`, to keep version-controlled maps.

### 3.4 Further Work

Our scheme works well, but it does not scale well to larger maps. Automating path generation would allow cartographers to handle only the rooms and hallways, with the paths being handled for them by software. Another option would be to let the computer define NavMeshes, which decompose the polygons in our map into convex shapes, and therefore, any path inside those polygons is valid. This would be easier with a different scheme (such as IndoorGML) since we do not have an innate sense of topology to the building and would have to employ computational geometry to find what polygons are ‘next-to’ each other. As it stands, polygons do not have to perfectly align, so this would be a significant challenge.

Another problem with our scheme is lack of software. We currently have to build maps using QGIS, which is a daunting piece of software. In the future, it would be useful to have an application that can handle maps with multiple floors, and perhaps making it easy to georeference and trace floor plans.



# Chapter 4

## Localisation

Localisation is the method of using contextual information, such as landmarks and known locations, to discover an approximate position of something. It is used across many industries and in many forms. Examples of localisation that can be observed throughout every day life are:

### Supermarkets

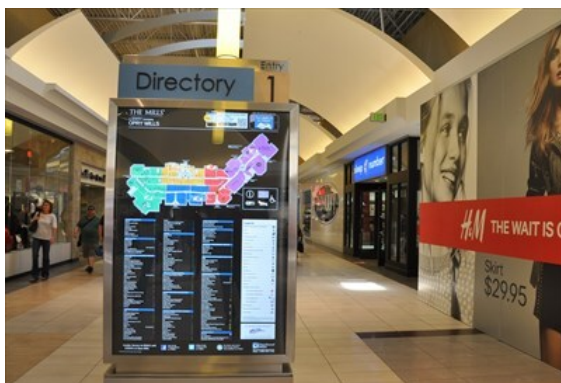
The use of signage above aisles telling a customer what category of products are stored there is a simple form of localisation. On their websites, supermarkets will sometimes tell a customer the aisle, and potentially the position, where a product is located.

### Warehouse management

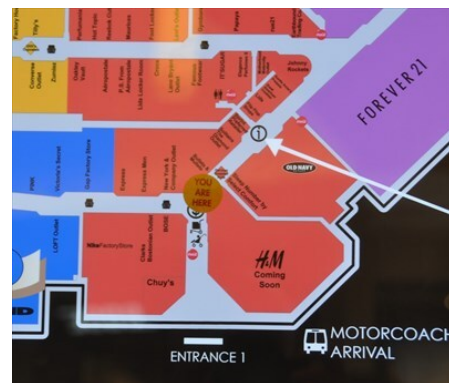
Warehouses use a form of localisation using a combination of a database, unique barcodes and QR codes. This information is used to allow the warehouse employees to look up a product and find its exact location for inspection and picking.

### Malls/Shopping centres

These have maps spread throughout the facility. These maps display their current location, usually marked with 'You are here' (see Figure 4.1b), and information about surrounding shops or other points of interest. This allows a user to find where a shop is in relation to where they currently are.



(a) A corridor in a mall with a map for visitors to use.



(b) Close-up of the map showing the users location and surrounding shops.

Figure 4.1: A map used for localisation in Opry mills mall (accessed from [7])

These methods are good in spaces that are designed with a single use in mind. For example, a warehouse exists solely to store and retrieve items in the most efficient way. Products of a similar category will be organised together in a way that makes their access efficient. This is a tightly controlled environment, and so the building can be designed around the use of the space.

In the Bragg building, we do not have this luxury, as it is a multi-use environment and is designed with the experience of its visitors in mind. This makes the building more complex to model. All of the examples above are environments where the localisation method is provided by the building owner. We are taking a more bottom-up approach, such that the localisation method does not require their cooperation.

It is also important to note that we are developing a human-focused method of localisation. For this reason a certain amount of error is acceptable. In a warehouse, robots may require millimeter-level accuracy, while the resolution of our system is in metres.

## 4.1 Existing Solutions

When outdoors, we can use the GPS system, which is the most commonly used localisation method in the world. However, walls, floors and roofs interfere with its signals, causing it to have much lower accuracy indoors. This is caused by the signals reflecting and refracting due to the construction of buildings, especially those built using materials such as steel. This changes their time of flight and hence drastically affects any calculations attempting to use them. Due to this interference, GPS is usually 5-20 meters off [29], which is infeasible for indoor navigation.

Indoor localisation is a difficult task. While some implementations exist, there is a distinct lack of universally viable and applicable solutions. The existing systems that are currently in use are in a very niche sector, and are unlikely to become widespread due to their high cost of implementation [18]. These implementations are sometimes called Indoor Positioning Systems (IPS).

### Dead Reckoning

This is a technique that uses sensor data to figure out direction and position, famous for use at sea. It relies on the knowledge of an initial prediction, and then using sensor information to update the location. This would be a compass heading on a ship, but in the context of mobile phones, it could use accelerometer data or other sensors. The main problem with this technique is that errors compound over time. Without a known location, the error in the position can increase rapidly. When we do know the location, we can set the error back to zero. This idea has been extended in Simultaneous Localisation And Mapping (SLAM) for robots, which uses sensor information to build a belief map of where the robot is. The belief map is then updated with new information from sensors [12].

### RTT

Wi-Fi Round Trip Time (RTT) is a new feature in the 802.11mc standard for Wi-Fi. This enables devices to measure the time of flight to a Wi-Fi access point and thus approximate the distance to it. The stated accuracy is less than 1 meter, which is perfect for our needs. However, this is a relatively new standard and there are very few devices supporting this feature [1]<sup>1</sup>.

---

<sup>1</sup>At time of writing, there are 4 routers that support this standard.

**5G** 5G mmWave technology provides angle-of-arrival measurements which can be used to *triangulate* the user. This has been used in China as an IPS in metro stations, where it achieved an accuracy of “3 to 5 m [...] in 90% of the platform and hall areas.”[16]

### OCR & QR

A potential idea would be to use various methods to display a variety of visual markers. This would fit into the larger field of computer vision - existing solutions utilise the user’s camera to match a captured image against a set of known images. These images would be tagged with locations, so finding a match would return the user’s location. This has the downside of being very intensive in terms of the amount of data that would need to be collected and is prone to errors in labelling. It would also be fairly computationally expensive and need to be offloaded from the phone itself.

### Barometric Pressure

Some smartphones are equipped with a barometric sensor, which can be used to determine the elevation of the user. Theoretically, the pressure would decrease as elevation increases.

### Magnetic Field Fluctuations

Another possible IPS uses magnetometers. takes into account of fluctuations in the Earth’s magnetic field over the space of a building. This is a fairly novel method, but has merit.

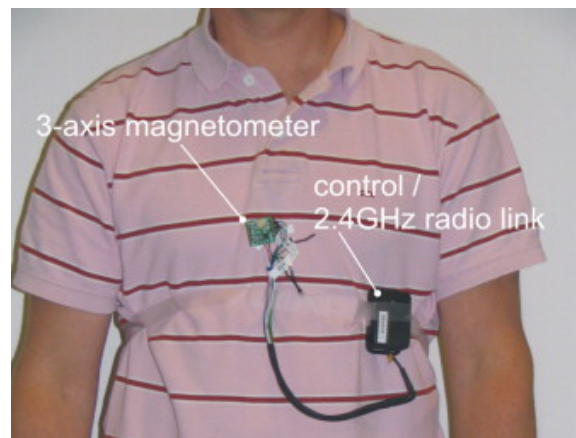


Figure 4.2: Magnetic field positioning (Reproduced from [15])

## 4.2 Possible Solutions

### Bluetooth Transmitters

By using Bluetooth transmitters, we could make use of Bluetooth Low Energy (BLE) Advertising Packets to localise the user. These packets are sent by BLE transmitters and received by mobile devices [17]. BLE Advertising Packets carry a lot of information. In our case, we would make use of its ability to send out its location and a connection frequency. From this information, a mobile device can derive its location.

### QR Codes

Quick Response codes work by encoding data into a 2D barcode [27]. Interestingly, localisation was the original use of QR codes, albeit using a different method. They were originally created to track vehicle parts along an assembly line during the 90s, with each part having its own QR code. This code was then scanned as it entered a new area [13].

Our method would use a similar concept. In our implementation, there would be multiple QR codes at key points in the building. The user would scan the QR code, transmitting the QR code's position to the app. This would then allow the app to show the user where they are.

### Wi-Fi Routers

Similar to Bluetooth receivers, Wi-Fi routers can be used to locate a person using the strength of transmitted signals, which is referred to as Received Signal Strength Indicator (RSSI). This can be used to calculate the distance of the receiver from the router. By combining signals from multiple routers with known locations, we can trilaterate the user's approximate location.

## 4.3 Proposed Solution

Our proposed solution should be straightforward to set up. Additional infrastructure being required would add significant overhead to the set-up of the solution, increasing its complexity and difficulty to maintain. In the case of QR codes, the building would require hundreds of QR codes to be distributed throughout it. The additional traffic these would create, due to visitors stopping to scan them, would be a concern.

With BLE receivers, the owner of the building would need to purchase tens, or even hundreds, of receivers and then spend time and money distributing them throughout the building. The time and money necessary to install Wi-Fi routers has already been committed, meaning the solution would make use of existing infrastructure.

By using Wi-Fi routers for localisation, we can take advantage of the infrastructure that is already in place by the university IT team. Therefore, we decided to base our solution on Wi-Fi router localisation. Our solution does not need additional hardware or infrastructure, and can be implemented at no extra cost. Most large buildings, such as university buildings and shopping malls, have numerous Wi-Fi routers distributed throughout the premises. As such, this solution will scale well to larger buildings, if they have sufficient Wi-Fi coverage.

Finally, we wanted the solution to be easy to operate. The use of Wi-Fi routers means that the application can automatically update itself, without requiring user input. Any changes to the building's layout can be immediately reflected on all client devices with no overhead. Most importantly, the solution would be on the user's own device; no stops to scan a code or other medium would be required.

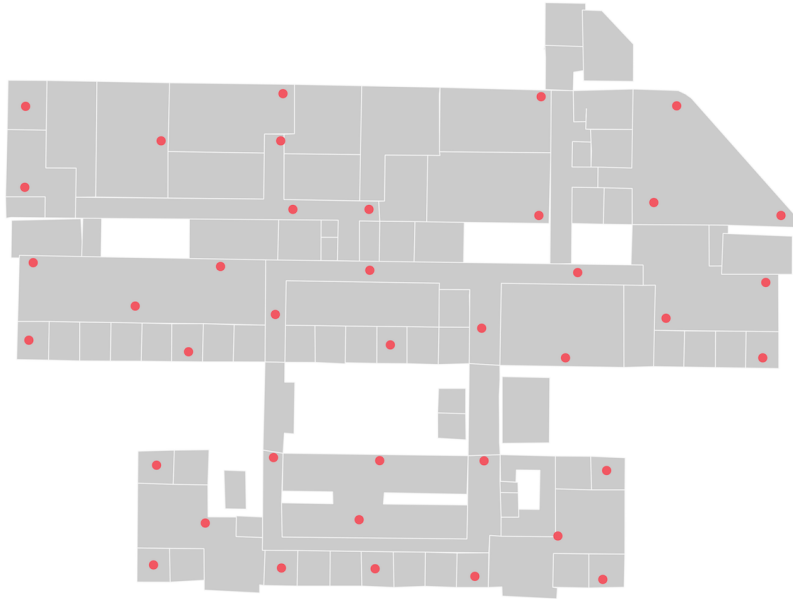


Figure 4.3: The Wi-Fi routers present on the second floor.

## 4.4 Implementation

For the solution, we created a prototype in Python and then implemented into a React Native application for use on mobile devices. The core library facilitating this process was Geodesy<sup>2</sup>/PyGeodesy<sup>3</sup>. This library contains functions and classes to aid the trilateration process.

### 4.4.1 Recording Router Information

We were able to obtain a floor plan of the building, which contained the location of every access point present across all floors<sup>4</sup>. This data also came with a list containing each router’s model, asset tag and unique MAC address (BSSID). In the case that a map of Wi-Fi router locations is not available, they can be manually mapped and located.

Using both the floor plan and list of access points, we were able to augment the QGIS map with Wi-Fi nodes. A node was placed on the location of each access point on the floor plan, recording its BSSID and unique ID. The SSID was discarded, as all routers broadcast multiple Wi-Fi networks with different MAC addresses (‘eduroam’, ‘MeetInLeeds’ and ‘WiFi Setup - University of Leeds’).

Any networks scanned by the mobile device can be cross-referenced against this data with its BSSID to discern which access point provided the signal. Since the routers broadcast multiple networks with slightly different BSSIDs, we compare the entire MAC address except the final digit, which changes depending on the network being broadcast.

<sup>2</sup><https://www.npmjs.com/package/geodesy>

<sup>3</sup><https://pypi.org/project/PyGeodesy/>

<sup>4</sup>A special thanks to Sam Wilson for the assistance in acquiring this data

<sup>5</sup>‘mac\_address’ is not an error, see Chapter 3

level	internet	ssid	mac_address
0	yes	wap70173	5c:a6:2d:ce:b1:a0
0	yes	wap70174	5c:a6:2d:cf:c4:a0
0	yes	wap70175	5c:a6:2d:ce:b8:20
0	yes	wap70176	ac:3a:67:08:26:20
0	yes	wap70177	ac:3a:67:08:06:20
0	yes	wap70178	5c:a6:2d:86:69:60
0	yes	wap70179	5c:a6:2d:cf:e0:60
0	yes	wap70180	ac:3a:67:07:fc:20
0	yes	wap70181	5c:a6:2d:cf:ab:a0
0	yes	wap70182	5c:a6:2d:af:8d:e0
0	yes	wap70183	ac:3a:67:08:32:40
0	yes	wap70184	5c:a6:2d:cf:c3:c0

Figure 4.4: A subset of the node data<sup>5</sup>

## 4.4.2 Trilateration

The localisation method we deemed to give the most accurate and consistent results was trilateration. The method of trilateration we implemented for this project made use of 3 distinct points and calculated distances in order to locate a user.

In this case, trilateration takes the user's distances to 3 known locations to approximate the user's location. In *triangulation*, we take the average position of the three inputs, while in *trilateration*, we use the distances from the points as well [31].

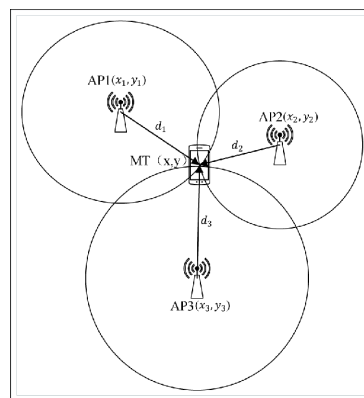


Figure 4.5: Illustration of trilateration (Reproduced from [31])

### 4.4.2.1 RSSI

RSSI stands for Received Signal Strength Indicator, which measures the quality of the connection between two devices over a wireless signal. The value is most commonly used to let a user know which nearby connections are the strongest, such as the strongest Wi-Fi signal in the local area.

RSSI is affected by many different environmental factors, including the physical makeup of the surroundings (e.g. steel, brick), temperature and humidity. An example of this is the fact that RSSI can be affected by just a small change in temperature and humidity. As can be seen in Guidara et al. [14], a slight change of 3.5deg Celsius can lower the signal strength of RSSI by as

much as 3dBm. An increase of 8% humidity can positively increase the signal strength by as much as 8dBm.

We used the following formula to calculate the distance from the Wi-Fi routers given the RSSI value:

$$\text{dist}(\text{RSSI}) = 10^{\frac{A-\text{RSSI}}{10n}}$$

This equation has two parameters:

$A$  RSSI strength when the receiver is 1m from an emitter

$n$  Accounts for all environmental factors that can affect signal strength

The  $A$  parameter depends on both the emitting and receiving device, though devices with the same chipset should give consistent results. For  $n$ , A major factor that affected the received RSSI values for our project was the sheer amount of steel used in constructing the Bragg building. There is no standardised way of working out  $n$  and so it is up to the person using the formula to calculate it themselves.

#### 4.4.2.2 Input Choice

To choose the input points for trilateration, we tested two different choices before settling on our final approach:

##### Three Strongest

Takes the 3 nodes with the strongest signal strength, which should be the closest routers. This would on paper be the best data to trilaterate the position of the user with.

However, this gave an error radius of 5-10 metres, which was too large to work in our proposed solution, especially considering 5 metres may be the length of a corridor.

##### Three Weakest

Takes the 3 nodes with the weakest signal strength, which should be the furthest routers. We theorised that by using the furthest away nodes the inaccuracies caused by the walls of the building would be of a similar magnitude to each other, as opposed to the closest where the signal from one node would pass through anywhere from zero to three walls.

Unfortunately, this was not the case; the inaccuracy when using the three weakest signals was similar to the three strongest, but had wild fluctuations where the predicted location was drastically incorrect.

#### 4.4.2.3 Chosen Method

In the end, we decided to try all possible combinations of nodes and take the average of all their predicted locations. This should reduce the effect of interference from walls through sheer data quantity, at the cost of a slight processing overhead.

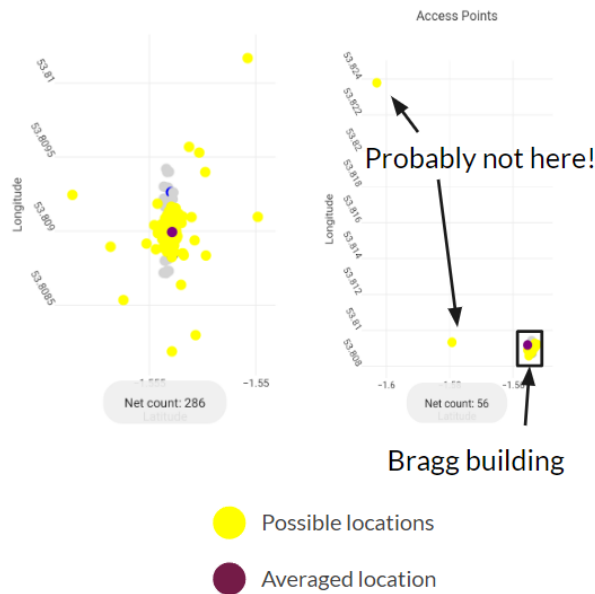


Figure 4.6: Two examples of trilaterated points and their average locations

This method proved to be far more accurate, but came with its own issues. During testing, the error radius would vary between 3 and 50 metres due to outliers.

To mitigate this, we added statistical filtering. We calculated the mean of all points and all data outside of two standard deviations from the mean was discarded. We repeated this process until either no points were removed, or the removal would lead to insufficient data to calculate a user's location.

The final solution was sometimes burdened by a significant amount of data, which exponentially grew the processing time of the application. In some areas of the building, the application would find upwards of 20 access points at any one time. Generating and processing all possible triplets of these nodes would create  $\binom{20}{3} = 1140$  possible locations for the user. Thankfully, the overhead from this large amount of data processing was negligible and caused no noticeable lag when testing the application.

Android's limitations of Wi-Fi scanning also impacted performance. Even when circumventing the four scans every two minutes limit, scans took around 5 seconds to complete, leading to all data being shown to the user being somewhat out of date. This was a hardware limitation that is beyond the scope of this project to try and circumvent.

## 4.5 Tuning the Parameters

Thus far, the parameters for the distance formula were chosen based on background research. These parameters assign  $A = 50\text{dBm}$ , and  $n = 3$ . However, these parameters require tuning to the specific building. Comparing the same formula and parameter pair across different buildings and locations will lead to varying degrees of accuracy. Walls, floors and the structural makeup of a building all affect the strength of Wi-Fi networks. The walls and floors of the Bragg building contain high amounts of steel, which causes mild interference [10].



Due to this, we decided to try and tune the parameters for the RSSI conversion formula, to get the most optimal setup for our chosen building. Initially, we had planned to tune the environmental factor,  $n$ ; most of the factors affecting the conversion were due to the building's construction, which is an environmental factor.  $A$  is derivable by taking scans at a fixed distance from an access point. However, we chose to tune  $A$  as well, for two reasons:

#### **Type of access point**

The Bragg building contains both generic Wi-Fi access points, as well as smaller extenders. Both are used in trilateration calculations, but the extenders would have a different  $A$  parameter, reducing the accuracy of the conversion.

#### **Better attribute combination**

There was a chance that we could find a better combination of  $A$  and  $n$  outside of the previously defined ranges, as  $A$  is not a parameter that is usually changed. This extended the scope of the task into an exploration of the best possible pair of parameters.

It is important to note that this tuning process was intended to match the execution of the application *to the maximum degree possible*. Any inaccuracies would lead to the results obtained being useless in the application. This concept informed a lot of decisions throughout the algorithm creation process.

### **4.5.1 Data Collection**

To facilitate this process, a large amount of data was collected from the entire Bragg building. We wanted our generated parameters to be as accurate as possible; even varying on a floor-by-floor basis, if deemed to be optimal by a measurable margin.

In total, we chose:

- 10 points for floors 0 to 3,
- 6 points for floor 4, and
- 3 points for floor 5.

The decreasing number of points is because floor 4 is a lot more closed off than previous floors (and we only had corridor access, rather than entering all the post-graduate rooms and labs), and because floor 5 consists of a single corridor and then the roof. These points were spread around the most accessible parts of each floor, normally on corridors or actively used rooms. This choice ensured the points covered the most commonly travelled areas on each floor.

A total of 10 scans were taken per point. This was facilitated by an access point scanner in the application, which now stored the detected access points in memory. A button was added to save all the recorded data to a file; however, this file was located in the root storage for the application which is not readily available. To circumvent this, the file would be copied to the phone's Downloads folder. This was accessible via a connected laptop, which could be used to copy the contents of the file into QGIS.

In QGIS, a new layer was added for these points. This layer stored all the provided data in a SQLite database, which was necessary due to the very large data strings provided from saving 10 scans at once<sup>6</sup>. A limitation was found where the data strings were too long for even QGIS to handle; they had to be manually added to the database, at times.



Figure 4.7: Map of scan locations on floor 3

### 4.5.2 Search Method

The idea to tune parameters was introduced late into the development cycle. Plans were being made at the time to finalise the user testing process and start to run the first tests with participants. The chosen solution had to be easy and quick to implement, in order to not divert resources away from more important tasks. Developing a complex, efficient algorithm would take time that we did not deem available. Hence, a simple solution was chosen; perform a grid search.

A grid search is a simple iteration over a search space with a uniform step size, forming a multi-dimensional grid upon which the value of the function can be evaluated and minimised. It is essentially a brute-force method, but for our example, where there are only two variables to explore and a small search space, it was easier to use than a more informed method like gradient descent.

This has the added benefit of creating a mesh of results, due to testing across a finite, complete set of values in two different dimensions. This mesh could then be passed into libraries such as Matplotlib in order to create graphs, creating a visual way to identify the best result(s) and find any patterns that arise.

---

<sup>6</sup>These strings would often exceed 65,000 characters.

### 4.5.3 Implementation

The tuning algorithm was written in Python using a Jupyter notebook. Python was chosen for this task due to ease of prototyping and the host of easy to access libraries to help facilitate this process.

The data from the SQLite database was loaded into the notebook. From this database, every point's network list had its relevant location mapped onto it. This MAC address mapping replicates the process the JS app uses before trilateration. As much code as possible was directly taken from the app solution, with slight syntax modification to make the code work in Python.

To perform the grid search, the NumPy library was used. This allowed a linear scale of numbers to be defined between two bounds, with a third parameter defining the number of divisions to make. Using two of these scales, a grid of values could be made and iterated through, facilitating the grid search process.

```
A = np.linspace(0, -60, A_div)
n = np.linspace(2.5, 5.5, n_div)
```

Figure 4.8: Creating a map via two linear spaces. The  $A$  linear space is bounded by 0 and -60, and will contain  $A\_div$  divisions. The  $n$  linear space is bounded by 2.5 and 5.5, and will contain  $n\_div$  divisions.

While developing the solution to this task, several unexpected issues arose that required solving. A brief overview of each major problem is as follows:

#### Insufficient nodes

In some cases, scanning for access points was unable to find a minimum of three nodes on that given floor. This was identified in locations that were at the very edge of the floor (normally by a set of stairs), surrounded by thicker walls and away from any major points of interest. This was a problem as trilateration requires at least three nodes to be performed. Without this minimum, the algorithm simply cannot be run. These data points were thus rendered useless and caused indexing errors until the root issue was identified. To handle this, the points with insufficient coverage were removed. Points with this problem were found on floor 3 and 5 of the building.

Latitude	Longitude	Scan Number									
		1	2	3	4	5	6	7	8	9	10
-1.55410325	53.8091697	19	17	17	19	22	19	19	19	20	19
-1.55392658	53.8091468	17	18	17	18	17	19	18	18	18	14
-1.55392563	53.8089408	11	17	16	15	17	18	17	15	15	16
-1.55417203	53.8089675	16	23	21	25	18	25	22	23	23	26
-1.55421388	53.8090438	17	19	19	17	20	18	19	17	18	19
-1.55420374	53.8087844	14	14	12	13	13	13	14	14	13	12
-1.55428850	53.8089485	16	15	15	14	14	15	16	15	14	15
-1.55416584	53.8092765	19	20	18	19	19	19	19	21	18	18
-1.55424165	53.8092384	12	12	14	13	15	15	15	15	15	15
-1.55434846	53.8092308	1	1	2	4	1	2	1	6	5	5

Figure 4.9: The points on the third floor of the Bragg building and the number of networks detected at each scan there. At the last location, a significantly lower number of access points had been scanned; trilateration is only possible on scan number 4, 8, 9 and 10.

### Floor 5 data starvation

The fifth floor of the Bragg building is still very much a work in progress. Only two staircases lead to this level; one of which leads to a very short corridor before the roof. The few rooms on the floor are still in construction and unusable.

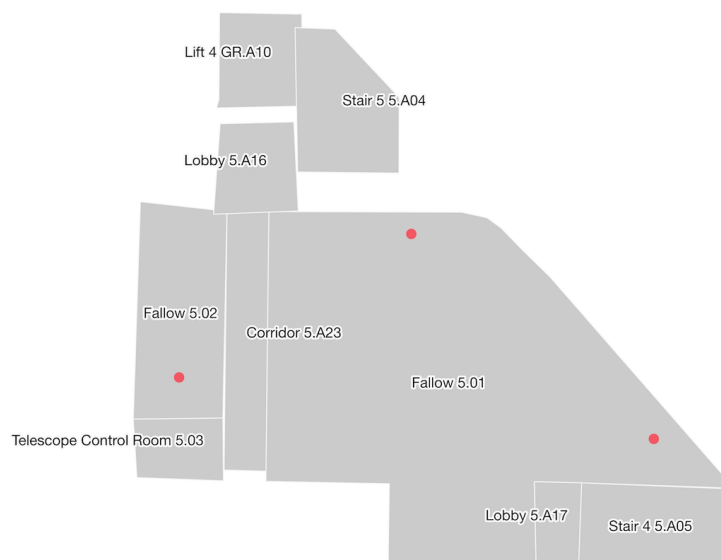


Figure 4.10: A map of the fifth floor, showing all three access points.

This floor has a total of three access points, located in the rooms and corridor off one flight of stairs. This means that unless all three are detected, trilateration cannot occur; if it does, the statistical analysis cannot be performed due to a lack of data. In addition to this, while detectable by an access point scanner, the nodes on this floor are not active, and cannot be connected to at the time of writing.

For this reason, this floor was not used for the final tuning solution, as any results gained would be of highly limited use. Furthermore, they would skew any processing of all floors cumulatively in a negative fashion.

After the removal of the unusable nodes and floor, we were left with:

- 10 points on floor 0, 1 and 2
- 9 points on floor 3
- 6 points on floor 4

Each point still had 10 scans performed at it, with sufficient nodes being obtained on each scan.

### Library language specifics

As mentioned previously, the trilateration solution hinges around the Geodesy library, which supplied helper functions and classes to significantly speed up the development of the algorithm. Without it, significant development time would have been spent on implementing functions that already exist, as well as debugging any inaccuracies that were added during the development process. Geodesy has a sister module for use in Python, called PyGeodesy. This allowed us to easily port the trilateration algorithm to the notebook, instead of having to re-implement the specifics manually and risk adding errors to the calculation.

However, upon testing the code, our tuning solution was exclusively returning errors on every single trilateration attempt. This initially displayed as a math error, but closer inspection identified the root cause to be an `IntersectionError`<sup>7</sup>, from PyGeodesy.

In the JS implementation used in the mobile app, a trilaterated point outside of the circle radii drawn from all three nodes is allowed; the predicted position is returned without issue. In the Python implementation, an error is instead thrown.

The Python implementation is technically correct. Trilateration requires the intersection of points to be accurate. We mitigated this with the sheer amount of data produced by trilaterating on every possible triplet, followed by refining the predictions using statistics. In addition to this, this tuning was performed late into the project; it would not be practical to rebuild and retest the trilateration algorithm with this new knowledge.

To combat this problem, we simply did not use the Python implementation of the trilateration function. We instead copied over the JS version directly and ensured the changes made so that it ran on Python did not impact the functionality or results. Using the function implemented in the application increased the accuracy of the tuning algorithm, due to directly using code from the main library instead of its sister implementation.

---

<sup>7</sup><https://mrjean1.github.io/PyGeodesy/docs/pygeodesy.errors.IntersectionError-class.html>

```

def trilaterate(point1, distance1, point2, distance2, point3, distance3, radius
=6371e3, point_nvectors=point_nvectors, Nvector=Nvector):
    # from en.wikipedia.org/wiki/Trilateration

    n1 = point1.toNvector()
    n2 = point2.toNvector()
    n3 = point3.toNvector()

    delta1 = distance1/radius
    delta2 = distance2/radius
    delta3 = distance3/radius

    # the following uses x,y coordinate system with origin at n1
    # x axis n1->n2

    # unit vector in x direction n1->n2
    X = n2.minus(n1).unit()
    # signed magnitude of x component of n1->n3
    i = eX.dot(n3.minus(n1))
    # unit vector in y direction
    eY = n3.minus(n1).minus(eX.times(i)).unit()
    # distance n1->n2
    d = n2.minus(n1).length
    # signed magnitude of y component of n1->n3
    j = eY.dot(n3.minus(n1))

    # x component of n1 -> intersection
    x = (delta1*delta1 - delta2*delta2 + d*d) / (2*d)
    # y component of n1 -> intersection
    y = (delta1*delta1 - delta3*delta3 + i*i + j*j) / (2*j) - x*i/j

    # note don't use z component; assume points at same height
    n = n1.plus(eX.times(x)).plus(eY.times(y))

    return Nvector(n.x, n.y, n.z).toLatLon()

```

Figure 4.11: The JS implementation of trilaterate, ported to Python. Some slight changes to comment layout were made for readability.

#### 4.5.4 Parallel Processing

Upon running the tuning process, we quickly came to the conclusion that the solution was unfeasibly slow. There are two primary reasons for this:

##### Dataset size

The sheer size of the dataset lead to processing taking a long time. In total, there were 450 scans used, each with a various number of networks detected. On average, approximately 15 networks were found on each scan, leading to  $450 * \binom{15}{3} = 450 * 455 = 204,750$  calls to the trilateration function for a single given  $A$  and  $n$ ; this magnitude of operations would take significant time to process.

##### PyGeodesy

Though it offers many functions that make trilateration easy to implement, the execution of methods using PyGeodesy is rather slow. They involve casting values into objects multiple times in a verbose manner, which adds up to a significant overhead over time.

To combat these issues, an attempt was made to pre-process some of the data. The trilateration function ported directly from the JS implementation involved creating several LatLon objects - one for each location passed to the function, for a total of three per point. These locations were used upwards of 20 times in a single scan in some cases, making the constant creation of these objects a significant use of resources.

A lookup dictionary was used to help reduce this overhead. Upon a LatLon object's creation, it was added to a dictionary, with the tuple of its location as the key. This dictionary could then be searched in future to see if the object already existed, avoiding the creation of a new object.

This helped, but not enough. The bulk of the time overhead was due to Vector classes embedded deep into the PyGeodesy call stack; rewriting and removing them would take a significant amount of time, and introduce many bugs that would risk the integrity of this tuning process.

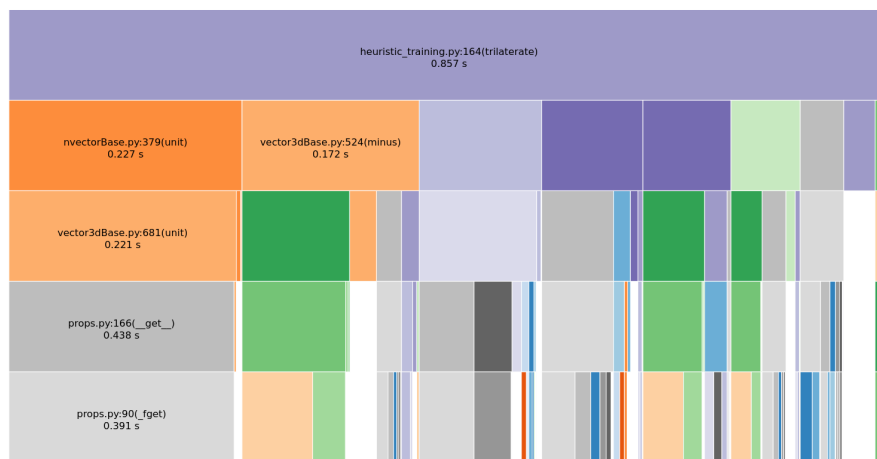


Figure 4.12: A profile of the main trilateration function using SnakeViz<sup>8</sup>. The majority of the execution time is locked in functions provided by PyGeodesy.

Pre-processing could only go so far. Python applications are subject to a Global Interpreter Lock (GIL)[26], preventing a Python application from executing multiple processing threads simultaneously. This stems from Python's memory access not being thread-safe; this behaviour is useful in most cases, but can be overwritten to increase execution speed by using the full set of logical cores on the machine.

Three different approaches were experimented with to try and increase the execution speed; threads, processes and a processing pool.

### Threads

These share the same memory as the global Python scope, but try to execute simultaneously in an attempt to run in parallel. However, this does not circumvent the GIL; instead, it just creates competition over processor time. An *increase* in execution time was observed when used, leading to this approach being discarded.

<sup>8</sup>A profiling tool for Python. [jiffyclub.github.io/snakeviz](https://github.com/jiffyclub/snakeviz)

## Processes

These differ from threads by bypassing the GIL, but each have their own private memory region. They can be executed in tandem, but incur an overhead by having to copy memory states from the global scope into each process. This overhead counteracts any gain from parallel execution, leading to the same execution time of 594s when tested.

## A Processing Pool

This is similar to using processes, but the user instead creates a *pool* of them to be polled from. The class itself splits the input data and assigns processes to each task, rather than this having to be defined manually [25]. Using this pool lead to a significant time reduction, taking 194s on a 10x10 grid search.

However, despite being by far the best solution, using the processing pool had an unusual teething issue.

Upon being used, the process spawned by the pool could not find any functions outside of the one it started in. The produced errors would repeatedly report that the function was missing from the declared scope, despite all being in the same global scope in the same file. This was not an issue caused by using a notebook - downloading into a Python file and running it caused the same issues.

To solve this, each function required for any other functions that it called to be passed in as arguments. This then allowed the function direct access to the scope in which they were declared, allowing the process to execute properly.

After all of the above, the tuning algorithm produced relevant and useful results, at a much faster pace than before; at the cost of stressing the machine we ran the solution on for a time.

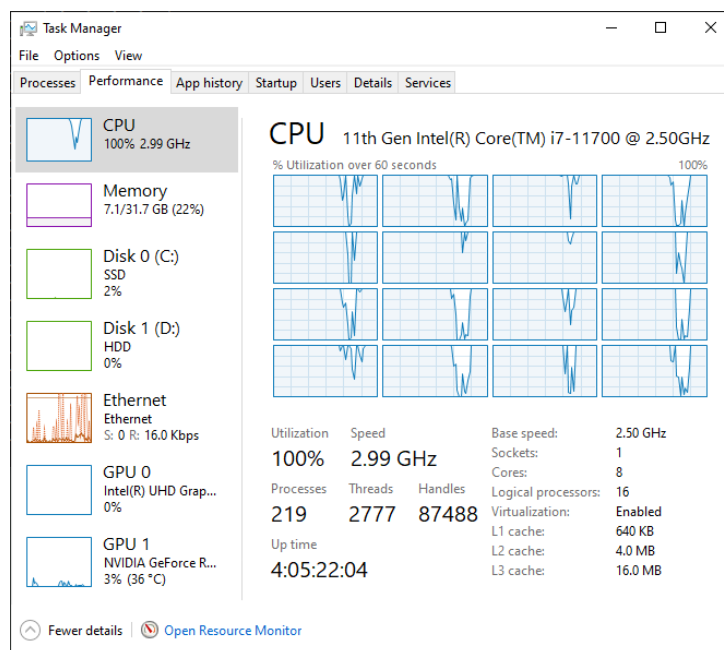


Figure 4.13: Resource usage by the computer used to run the parameter tuning. The CPU would be running at 100% for multiple hours at a time.



### 4.5.5 Outcomes

As discussed previously, the lack of data available on the fifth floor, due to exceptionally poor Wi-Fi router coverage, led to any data obtained being either unable to be processed, or being the cause of inaccurate results. Due to the possibility of any values obtained skewing the overall building results dramatically, it was stricken from the optimisation; hence no results for it were obtained.

The tuning algorithm was ran on each floor individually, as well as upon the entire building at once. Initially, scans were taken over a range similar to the base parameters; in the range of -30 to -70dBm for  $A$ , and 2 to 4 for  $n$ . However, upon testing this, the best results obtained were always on the edge of the grid bounded by these values. Furthermore, the same ‘trench’ pattern was present across all floors and subsets of data; this was not some coincidence or one-off occurrence.

The search area was expanded repeatedly, until far out of the usual scope for these values. This was evidence that a better combination of parameters did exist, that was likely bespoke to this building. The final search area ranged from 64 to -64dBm for  $A$ , and from 2 to 8 for  $n$ . To account for this wide search area, two scans were executed on each data set - a scan over the entire area, followed by a scan around the best point located. This second scan had a search area with a delta between maximum and minimum value of 20dBm for  $A$  and 2 for  $n$ . The search area for each produced graph will be noted next to it.

Data obtained by running the algorithm on each floor individually can be found in Appendix C. For the purpose of visualisation, the error on each graph is capped to 10m. Without this cap, certain parameter combinations would lead to an error of upwards of 200m, making any patterns difficult to observe due to the adjusted scale. The darker the colour of the mesh, the lower the error is.

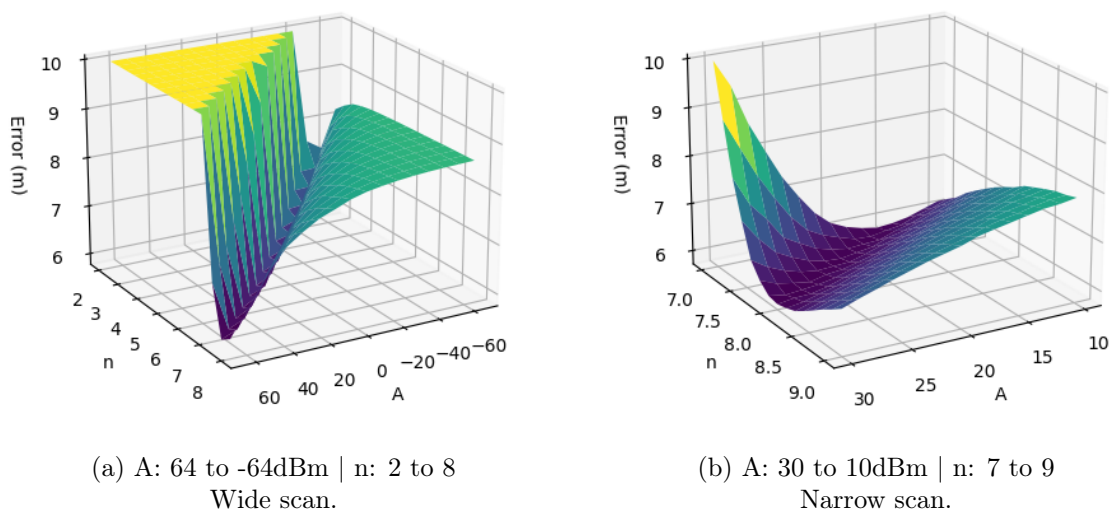


Figure 4.14: Graphs produced from the Wi-Fi scan data for the entire Bragg building.

The best parameters obtained for each floor were:

Floor	A (dBm)	n
Ground	37.00	8.36
1	25.57	7.71
2	28.00	8.71
3	19.86	7.07
4	19.86	7.07
All	25.71	7.71

The new parameters of  $A$ : 25.71dBm,  $n$ : 7.71 were implemented into the app for evaluation in user testing.

To test these new parameters, we recorded the error obtained at every point used in the optimisation process. This could then be compared to the error when using the old parameters ( $A$ : -50dBm;  $n$ : 3), in order to draw conclusions about its effectiveness.

These tables can be found in Appendix D. Every entry is the average distance (in metres) between the calculated location of the user and the actual location, for that specific point.

For ease of comparison, below is a table directly comparing the error of the new parameters to the old set.

Point	Floor					Average
	Ground	1	2	3	4	
1	-1.16	-0.26	-1.59	-1.62	-5.85	-2.09
2	-1.81	-4.22	-1.04	-4.33	1.30	-2.02
3	-0.98	-2.67	-1.34	-4.41	-4.00	-2.68
4	-1.53	-3.06	-2.54	-1.91	-1.40	-2.08
5	-6.81	-2.08	0.64	1.24	-2.46	-1.89
6	-3.62	-3.81	-0.53	-2.83	-1.23	-2.40
7	-4.17	-4.26	5.09	-6.18		-2.38
8	5.70	-2.62	-0.31	-2.89		0.10
9	-1.98	-4.40	1.77	-1.10		-1.43
10	-3.51	3.15	-2.36			-0.91
Average	-1.99	-2.43	-0.16	-2.68	-2.2	<b>-1.81</b>
Std. Dev.	1.59	0.01	1.28	-0.89	-1.69	<b>0.44</b>

Figure 4.15: The performance of the new set of parameters, compared to the original set. An increase in performance is marked in green; a decrease is marked in red.

Over all the data points, the average error distance decreased by -1.81m. This is supported by the general distance metrics in the table; the majority of comparisons had the new parameters cause a notable improvement. There do exist outliers however, which are sometimes a drastic increase. An example of this is point 8 on the ground floor; +5.70m is a significant change, skewing the standard deviation. In general, the standard deviation when using the new

parameters did increase. This implies that while localisation is more accurate, any inaccuracies are more pronounced than before.

As observed in Appendix D, we obtained an average error of between 4 and 6 meters. The overall averages were skewed by some significant outliers, which can be attested to data starvation in remote parts of the Bragg building.

Though this is not a measurable metric, it is worth noting that the app did feel more accurate after tuning. The predicted location was more accurate and reactive when the mobile device was moved; transitions between rooms were represented more accurately than before. Furthermore, the app reacted less dramatically when stairs were taken, being able to locate the user in fewer scans than previously. We were not able to test this with user surveys; this was only via anecdotal evidence whilst developing.

## 4.6 Future Work

### Path snapping

If it can be proven that the path produced by the application is optimal, the predicted location could be ‘snapped’ onto this path. This operates under the assumption that the user does travel along this optimal path. Instead of being a true localised position within a building, this would instead represent a ‘progress’ metric of travel along this optimum. However, this would pose issues if the user did not follow the path; be it via user error or an event such as the way being blocked. In such a case, the user straying would leave them without information on where they are located, leading to them becoming lost.

### Weighted prediction heuristics

Most navigation through a building like Bragg is performed via corridors. Instead of snapping to a path, more weight could be given to predicted locations that lie within corridor *regions*. This is more likely to give a useful user location, as they *should not* be navigating through a series of rooms to get to their end goal. This would add some processing overhead to test what region a predicted location lies in however.

### Data starvation testing

Bragg is a building with strong Wi-Fi coverage. Removing a proportion of the test dataset could provide information on how the current parameters work with less data, thus giving insight as to how the app would operate in a building with far fewer access points.

### Wi-Fi RTT

When the Wi-Fi 802.11mc standard is more widely supported, it could be used for indoor localisation. It may perform better than our RSSI-based method and may be faster. This standard is similar to how GPS works and would work as an ‘indoor GPS’.

# Chapter 5

## Server

### 5.1 Architecture

The overall architecture of the server is fairly standard, there is a database that talks to the API which talks to the client. We aimed for the server to be as agnostic as possible to the format of the map, this is because there is more flexibility in the client’s representation of the map than the server.

To achieve this we use only the most basic elements of a feature (e.g. geometry, IDs, etc.) and leave all the other data in a ‘tags’ field. There are a few places that implement strictly optional behaviours if those elements are present, the pathfinding for example weights room-to-room pathing if it can as this can create paths that might be disruptive to people in the building; we don’t want people wandering through offices.

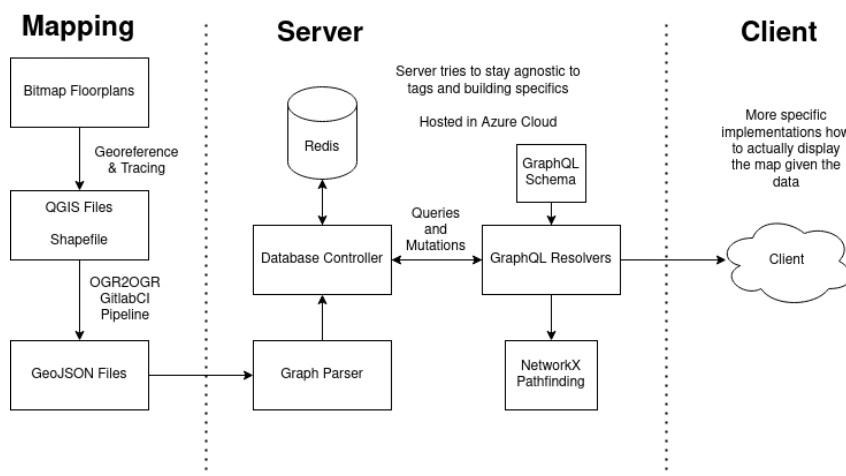


Figure 5.1: Diagram of server architecture & map data processing

This leads to a fairly robust and open-ended server that can deal with changes both from the client and mapping end, the server is more loosely coupled than if we were to agree on a schema that all applications in our ‘stack’ should follow. This also means that if a mapper can’t find out specific pieces of information about certain map elements, they shouldn’t worry about how the server (which is furthest from their control) will deal with it; this is then up to the client.

### 5.1.1 Parser

A major part of the server is the ability to read GeoJSON and turn it into a suitable data format for our purposes.

Some challenges were raised in the fact that the path was represented as a LineString feature, we took each of the points in the LineString and converted them to the PathNode dataclass. To de-duplicate nodes we took the coordinate tuple and used that as a lookup in a python dictionary, this assumes that two points with the same coordinates are the same point, which is a not-unreasonable assumption to make, however this might have its downsides if specific mapping results were desirable.

Another point that required work was the check for if a node is in a given polygon. At first, we were using the `PyGeodesy` library, this has a check to see if a given point is in a polygon, however it is very slow. To get around this we wrote a bounding box check that only called the `PyGeodesy` check to resolve where there were more than one bounding boxes that were intersected. This too was complex and slow, so we decided to ignore the curvature of the earth and use the `Shapely` library whose geometry types were much less expensive to instantiate and polygon bounds checking was much faster overall. This does have the drawback that if our system was used to map a building on the North or South Pole it may not work as intended (it's hard to put a precise latitude number on when this assumption becomes unworkable). Overall it is fairly easy to swap out the libraries here, so there is little concern over this.

This is also the stage where edges between each node in staircases and lifts are created, this is a weak point of the scheme we are using as it isn't encoded in the map itself, however a GeoJSON file could be produced after parsing that did encode this information; and in-fact the app recreates a GeoJSON format to render geometry.

We used this intermediary format because we needed to create the graph in a way that python would understand. This also made it easier to store in the graph format in the database. A totally valid way of approaching this problem would be to not use a parser and serve the GeoJSON files directly, just using a regular relational database (e.g. MySQL or Postgres) or a document database to store them. This would be the best way if we didn't have to process the map further after QGIS creation, in the future if software was created that let us create the final GeoJSON directly this would be a good solution to explore.

### 5.1.2 Database

We chose to use Redis with the graph and search extensions for the database part of the server application. The RedisGraph extension fit the structure of the mapping data that we chose, with relations between nodes that have data attached to them. This works with the RediSearch module to allow full-text search over properties in the graphs to be very easily developed.

We didn't take full advantage of the graph database portion of RedisGraph as not everything is included with relations due to limitations of how the full-text search with RediSearch works; ideally polygons and path nodes would have a one-way 'inside' relation between them and the same with PoIs and path nodes having a 'closest' relation.

Currently, these have a prefix of the graph they should be in along the lines of ‘<graph\_name>:polygon:’ which we can therefore scan the database for and split on the colon character.

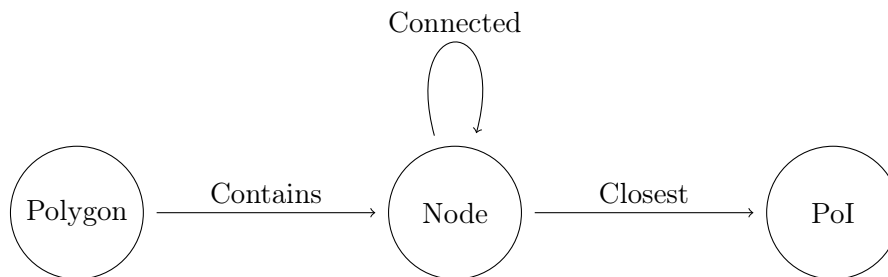


Figure 5.2: Data structure of graph

Ultimately for our small application this doesn’t lead to many possible queries that are hit by this, the biggest being the query for nodes within a given room (using search). Also, the returning of all polygons and PoIs suffer from this. In the future we could include all of these pieces of data in one graph.

There are some overall problems with scalability using RedisGraph as it cannot be clustered like a regular Redis database, however different graphs can be distributed across different database hosts. To implement this we could have a master or clustered Redis database that contains keys that correspond to each graph and an address of the database host they are stored on.

### 5.1.3 API

The API that the client application uses to access the server uses GraphQL, we chose this over a REST API since it allowed prototyping the app faster, and has the added benefit for a much more flexible API whilst developing. GraphQL allows you to define types and queries that return those types (or some collection of primitive types). By doing this the control over the specifics of the query are handed off to the client and the server just exposes the data and the structure of the data that it is willing to send to the client.

This GraphQL API was implemented using the Ariadne library, for which you have to write a series of resolvers for each of the types you define in the GraphQL schema (see Appendix E). There are also resolvers for queries, these contain little logic and defer to the database controller in most cases.

Each of these types have properties similar to a class in Object-oriented programming, by allowing a query to just the properties that client needs we can save on both processing time and bandwidth. For example in the `Node` type there is a `polygon` field that corresponds to the polygon that the node is contained in. The query for this involves looking up the polygon by id and thus has a higher latency than if we didn’t request it. Similarly the `levels` field in the `Path` type must loop over all of the nodes in a path to find the levels that the path spans, if we don’t need that data we can simply not process it.

Most queries simply return a specific by-id piece of data, or a list of all the available data. This means that we can save on the amount of data sent if for example we ask only for the nodes' IDs and the edges and only look up data about the specific nodes along the path we are travelling, whilst this isn't a concern for the moderately sized Bragg building, it could help in larger buildings, or in applications where this mapping scheme might store documents alongside a room for example.

Search queries simply take a single search string, but there could be improvements in the future to allow for options to be passed to Redisearch for different search options in client applications. Some improvements were made to the default search, notably increasing the maximum number of results from 10 to 20 which allows more general searches e.g. 'office' to work more effectively.

A scalar type for tags was also written, this is just a type that returns a python dictionary as a JSON object that Ariadne can understand and return properly for the client.

These queries were mostly written in anticipation for the data that the client would need to operate, for this reason some changes needed to be made after the initial application was written but for the reasons outlined above this was very fast.

#### 5.1.4 Path-finding

Server-side path-finding is completed by passing a set of nodes and edges into a class that uses NetworkX to run A\* with a heuristic function that tries to stop paths being generated that are undesirable (for example paths that run between rooms when a hallway is available). Overall this class is fairly simple, some attempt was made at generating text instructions for a given path, but we decided it would be better for client applications to do this as they could implement it using their own idioms.

We discovered in our testing that there were some bugs with pathing around staircases in rare instances it would direct the user to go downstairs then back up again. This is because our heuristic function adds no extra weight for staircases and lifts. In the future this will have to be explored and tuned to make more realistic paths.

It should be noted that the paths we are trying to generate are not 'optimal' in terms of distance but instead are trying to fit to a perceived convenience and accuracy to a path that someone would actually take. Paths that take you through an active office might be optimal in time and distance on paper but would actually involve more time to navigate as you would probably have to apologise to everyone in it!

## 5.2 Build & Deploy

To build the application we decided on using 'Pants' which has the ability to package an application as a .pex binary that packs all the dependencies into one binary. This pex file was then built into a docker image using a Docker file which, was uploaded to GitLab's container repository by the CD pipeline. Docker-compose could then be used to deploy the app to Azure

web-apps, this was done automatically by defining a web-hook so that on any event that caused a new docker image to be built Azure would pull it and restart the server.

Listing 5.1: Dockerfile

```
FROM python:3.8
ENTRYPOINT ["/bin/mapping-app"]
COPY server.src/app.pex /bin/mapping-app
EXPOSE 80
```

Listing 5.2: docker-compose.yml

```
version: '3.7'
services:
  redisgraph:
    image: redislabs/redismod
    container_name: redisg
    ports:
      - "6379:6379"
    restart: unless-stopped
    networks:
      - redisgn
    hostname: redis
    volumes:
      - redis:/data
  mappingserver:
    image: registry.gitlab.com/comp5530m-mapping-project/
      comp5530m_mapping_project/mapping-app:main
    container_name: mapping
    ports:
      - "80:80"
    restart: unless-stopped
    networks:
      - redisgn
networks:
  redisgn:
volumes:
  redis:
```

Builds are completed using GitLab CI with docker-in-docker to build the docker image and pushed to the GitLab container repository where then a webhook is triggered telling Azure to pull the new image. This is a fairly flexible system for deployment to the single Azure instance but would need to be more complex if we were running multiple servers (even if they were all mirrored). Builds are completed if there is a change to the server code or any of the build config files on any branch, and the images are tagged with the branch name, this means that we don't have to merge to main to test out features. To use another branch you can then simply change the 'image' line in under 'mappingserver' to point to the branch tag.



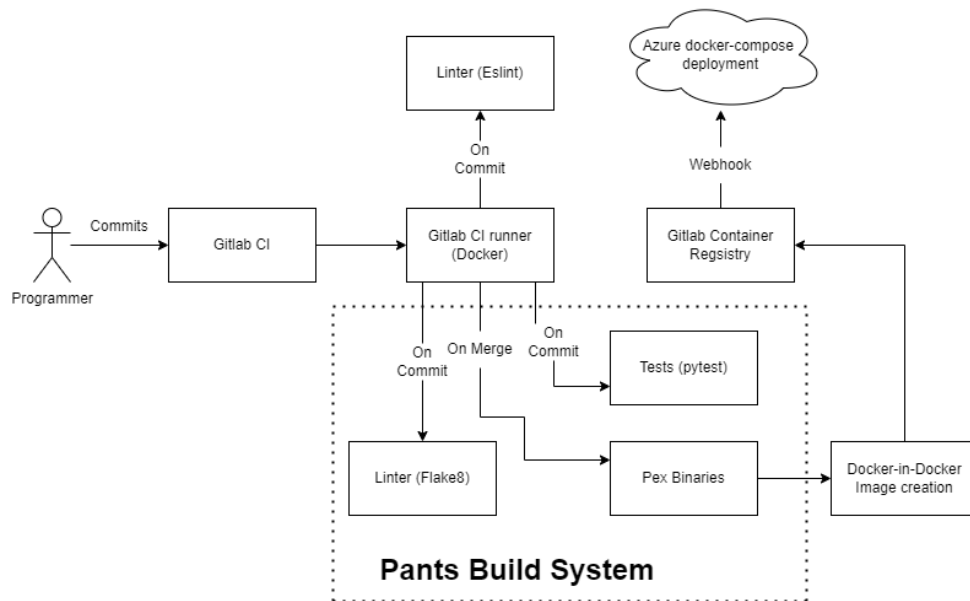


Figure 5.3: Diagram of CI/CD process

We also ran tests on the server, and lint on the server and the app to ensure we had nicely formatted code, and that there weren't any regressions with server changes. These were handled using pants for python and ESLint for JavaScript.

Since we used the Azure free tier for development, there were some problems with the server being de-provisioned whilst we were working on the app (especially if there had been no activity for a while). We also ran into issues with usage limits if we had been developing the client app and were refreshing a lot. Both of these issues were solved by moving to the paid tier during user testing, but were a concern.

### 5.3 Further Work

As mentioned above, it would be good to have everything in the RedisGraph structure itself, this would mean that we could make queries in general faster if there were many buildings in the database. I think this is the major limitation of the scheme that is currently implemented, but it is not slow.

Another issue with the system is its overall scalability, this is not only because of RedisGraph's own limitations, but the fact that there are no mechanisms that could load balance the API and the database components separately from each other. At such a small scale this has no real impact but if this application was used globally we may want to look at something like a microservice-based architecture. This isn't without its problems however as there is only really one domain that the database can refer to meaning we'd have to figure out where the data should be stored and cached (which would be a problem that had to take into account the location of data centres etc.).

Finally, it would have been nice to implement a way of generating paths, as outlined in the conclusion of Chapter 3. This would have made the parse more complex and slow, which we

were trying to avoid and ended up just being out of the scope of the project.

Finally, we could improve server by adding a query for nearest buildings given the current GPS location. This would allow us to load building maps before a user enters the building. This could be used to load a map before the user was inside the building, allowing for multiple buildings to have been mapped and tested. This functionality would use Redis' geospatial abilities to query a radius around a coordinate and would involve adding a polygon or point in the building that could be added to the database to query against.

# Chapter 6

## App

### 6.1 Implementation

Initially, we wanted to create a web application. Web apps can be used across platforms and do not require installation on the user's device. However, after deciding to use Wi-Fi information as our localisation method, we found that web applications can not access network information. There is currently an experimental API which measures network quality [20], but it does not return the RSSI values necessary for localisation. Therefore, we chose to write the client as a native application, which can access the RSSI directly from the operating system.

We chose to implement the application in React Native, which is a JavaScript framework for writing native applications for Android and iOS. React Native is also cross-platform, allowing the application logic to be written once and compiled for each operating system; it also allows us to bundle other JavaScript libraries to add functionality e.g. drawing vector graphics or querying a GraphQL server.

### 6.2 Design

#### 6.2.1 Technical

React Native uses the React UI library to structure the application. Each 'screen' of the application is made up of multiple nested components, defined by the JSX format. They can be functional components, or defined by classes. Components specify the data they depend on and how the data is displayed, and React orchestrates the creation, modification and deletion of components as the data changes.

An example of a component is a button, these buttons can be composed together by other parent components, much like HTML.

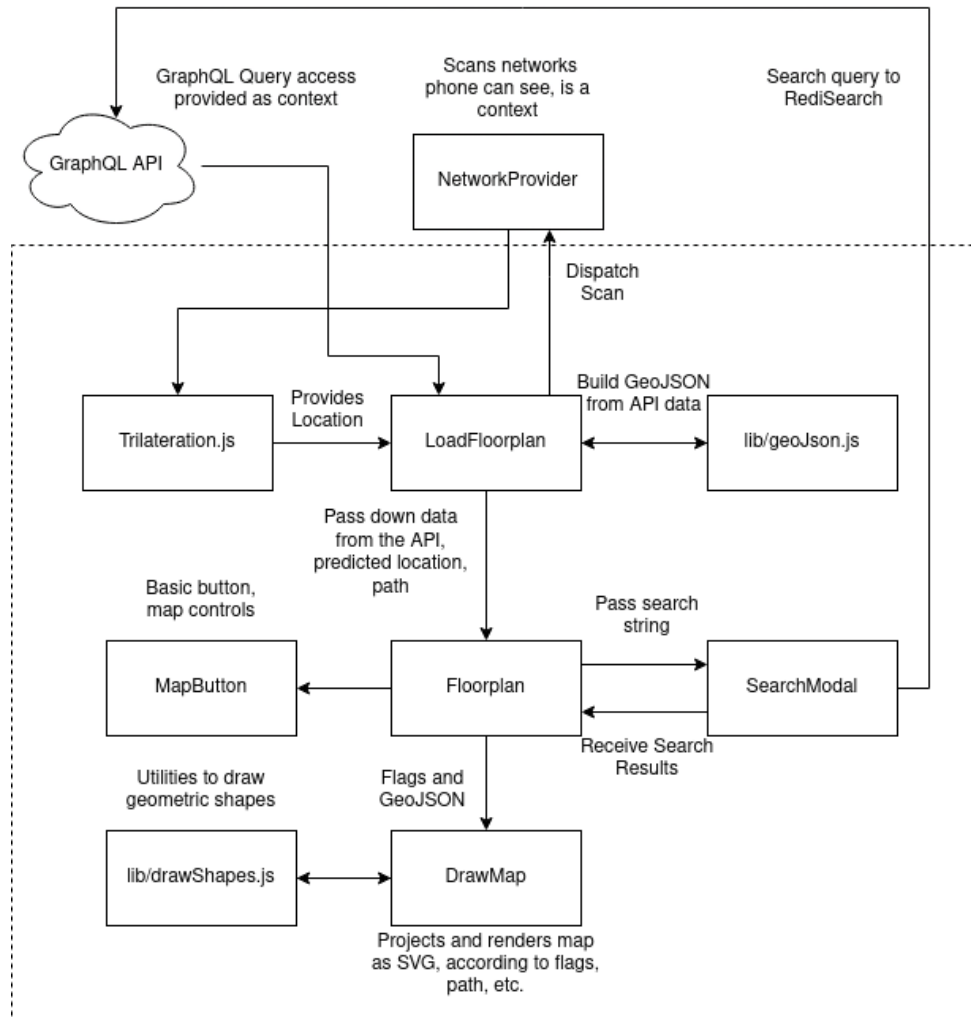


Figure 6.1: Architecture of the app by component

### 6.2.1.1 Client-Server Communication

When loading the initial map or requesting a path to a room, the app needs to contact the server and retrieve the appropriate data in the correct format. We use Apollo [3], a GraphQL implementation for JavaScript, to write and execute queries.

### 6.2.1.2 Network Information

As mentioned in Chapter 5, the localisation relies on the strengths of Wi-Fi signal between a device with app installed and nearby Wi-Fi routers. Devices with Android 9 onward have Wi-Fi throttling enabled by default, which limits the Wi-Fi network scanning to four times every two minutes [2]. Worldwide market share of Android devices using version 9 or newer is over 70% as of April 2022. [30]. For our localisation to provide real-time data, we need to scan the networks every few seconds. Therefore, for the app to work as intended, the devices with the app must open developer settings and disable Wi-Fi throttling.

## 6.2.2 UI / UX

We aimed to recreate the design and functionality of other successful outdoor map apps, such as OsmAnd, Waze and Google Maps. These all use a fairly standard design language and as such have created an expectation from users as to how a map app should behave and even where certain buttons should be. For this reason the problem of creating such an app is fairly constrained, and we should strive to repeat successful examples of such.

The most obvious is the map itself. The map should support pan and zoom operations at a very minimum, a lot of apps also support the ability to rotate the map. Panning and zooming is core to being able to contextualise where you are in the map.

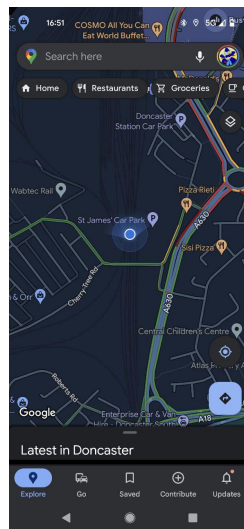


Figure 6.2: Google Maps interface

The map is visualised as a two-dimensional floor plan with different colour schemes for rooms and corridors. The location of the device is displayed as a marker, and the starting and target location is shown with a distinct colour. Additionally, the map can optionally display room

labels and points of interest. This visualisation is generated with a third-party JavaScript library d3: a library for producing dynamic and interactive data visualisations.

The map is drawn using SVG, this was a good solution as it allowed us to use d3's geo libraries to project polygons to an SVG path easily. SVG is a simple and fairly light-weight solution but doesn't work well with React-Native's refresh and re-render strategy. For this reason picking a more specialised graphics library might have been better for performance; SVG did however work well in the end and allowed us to create a final product quickly. Another potential optimisation here would be to allow the SVG to be 'pre-calculated' then save it to memory, loading it back if it's already been created. This would avoid re-calculating each element's projection from latitude and longitude to the screen coordinate paths. This pre-calculation could only be done for the static elements of the map.

### 6.2.2.1 Location

The location is represented as a circular marker, this has a translucent circle around it representing error. It is updated every time the access points are scanned, which takes five seconds; this is a process that cannot be sped up as it depends on the radio in the phone. We therefore use the accelerometer and compass to provide an estimated location between scans, represented by a triangle shaped marker, this works as a sort of basic pedestrian dead reckoning.

If the user is moving (as detected by a weighted sum of accelerometer readings) the triangular marker will move a set amount in the direction of the compass heading reported by the phone. This ends up in a sometimes wildly inaccurate prediction, but we felt that it helped let the user know that the app was doing something between the full Wi-Fi scans as otherwise there was no feedback.

Inspired by Google Maps, after unsuccessful network scan the location marker turns grey to alert the user that location data could not be obtained.

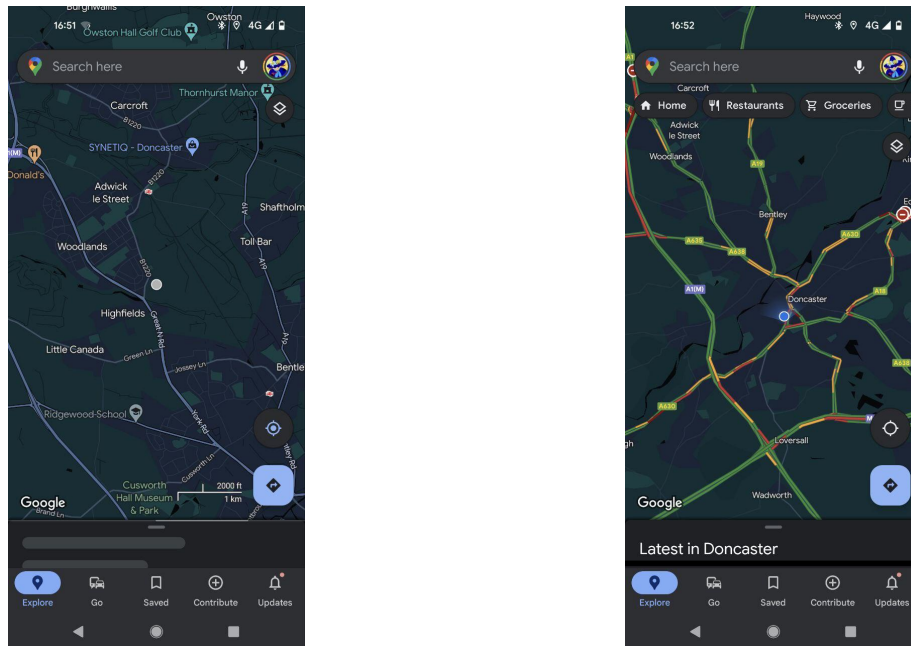
### 6.2.2.2 Search

Search is handled by sending requests to the server on each character change, the app dispatches a request and shows the results. The search bar is at the top of the screen, and opens a modal over the map window to select the room to navigate to.

### 6.2.2.3 Floor Changes

Since other apps do not have this idiom, we needed a way to handle floor changes. Ideally the user would be able to look ahead and see what was the floors above and below where they were, but also the app should change floors automatically when they reach the next one.

Initially we had a toggle between a sort of 'free browse' mode and a 'follow mode' similar to how Google Map's button works (see bottom right of Figure 6.3b). But decided that this was not communicating the idea properly.



(a) Google Maps displays a grey dot when using the old location      (b) Google Maps displays a blue dot when it has a fresh location

Figure 6.3: Google Maps location markers

We worked out that we could actually remove the toggle and enable a follow flag if the user was looking at the floor they were currently on, if they switched floors to look ahead this flag was set to false. This meant that if they switched to the floor they were looking ahead to the flag would be re-enabled and follow them again. This took a button away from the user interface, simplifying interaction.

As an example if the user was on floor 1 and wanted to see what was ahead on floor 2 they would press the button to move one floor up, setting the flag to false. If the user walked up the stairs to floor 2, the app would locate them and set the flag to true. The user could then walk up the stairs again and the app would follow them to floor 3.

This made sense with how people would use the app, the toggle in Google Maps makes sense as the map is significantly larger and enables a zoom and pan to the user's current location. Our map does not need this as the whole map fits onto a phone screen easily.

#### 6.2.2.4 Other Affordances

Visual feedback was key when developing the application, but we also decided to add vibration feedback when pressing buttons and importantly when a new location update was received. This was intended to let users know that the app had updated the predicted position even if they were not currently looking at the screen. We also added vibration when pressing the buttons in the app, just to let people know they were received.

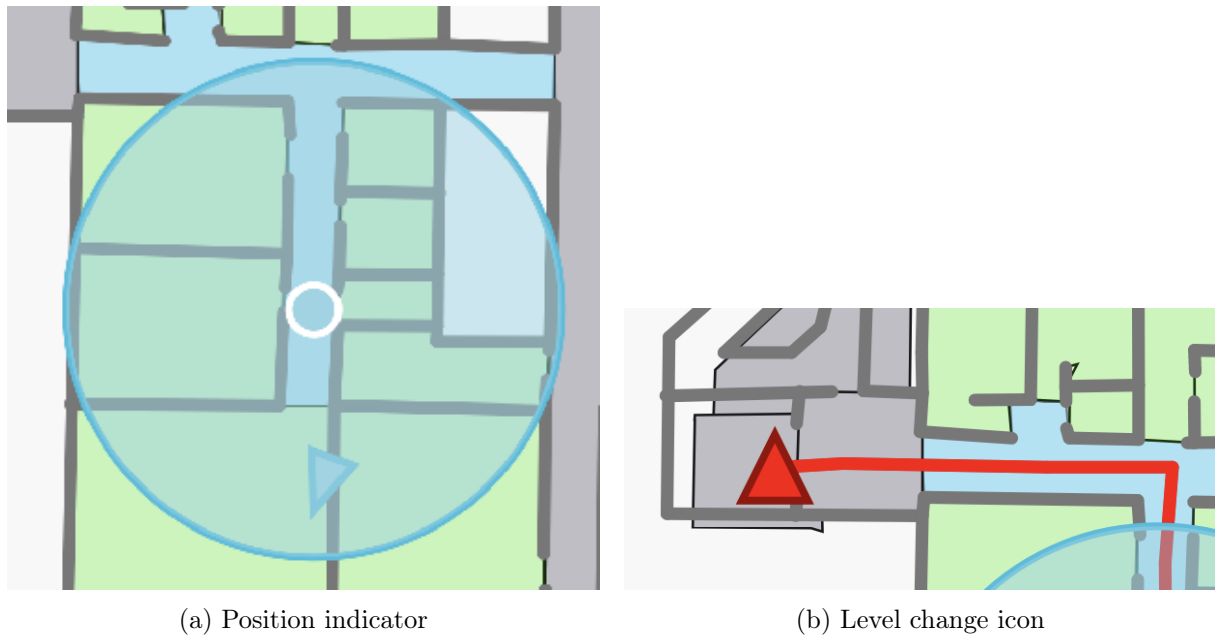


Figure 6.4: UI affordances for navigation

### 6.3 Design Iterations

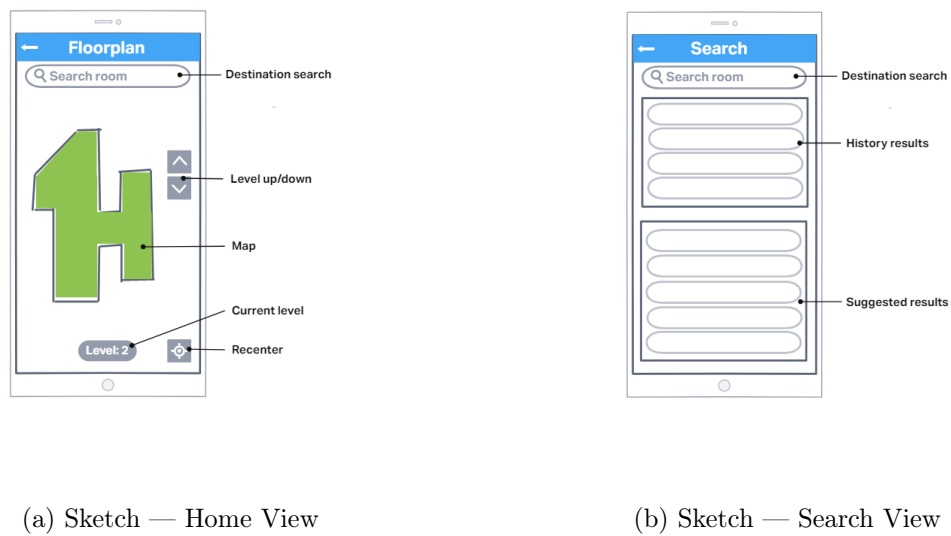
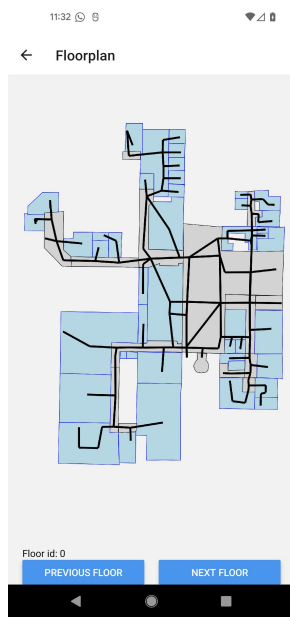


Figure 6.5: User Interface Sketches

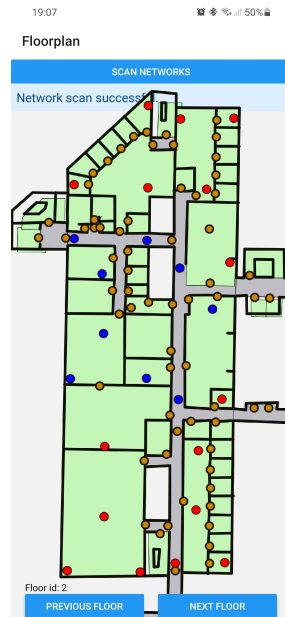
The app design was initially drafted with hand-drawn and digital sketches. The most important aspect of the design was to make it simple and usable without any prior app exposure.

Later, when the application was developed, the design was going through multiple iterations. We have been experimenting with various ways of room searching, button placements, and colour schemes until we achieved a pleasant and intuitive user interface. Throughout the iterations, we also graphically represented a lot of debugging data, that we decided to not include in the final release version.

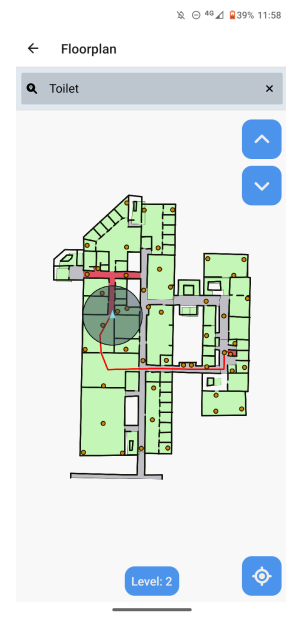




(a) Design 26/02/2022



(b) Design 04/03/2022



(c) Design 15/03/2022

Figure 6.6: User Interface Iterations

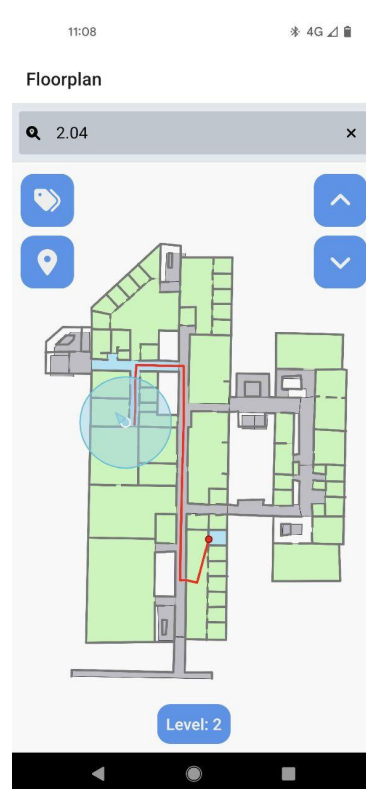


Figure 6.7: Final design

## 6.4 Further Work

The application lacks client-side support. If a user has any issues or improvement ideas, there is nowhere to report them at the moment. There is currently no need for client support as the app is not meant to be released publicly, and we collect all the feedback through user testing and forms. Although, if the app was about to be released, we would have to introduce a way to provide support.

Map data is downloaded from the server every time the app is started on the user device. This approach is very inefficient as users are likely to need the same map multiple times. In future development, the app would benefit from data caching. It would not only speed up app startup but also limit server usage. Additionally, path-finding could also be done on the client-side, rather than on server.

The current map visualisation solution works, but is very limited. Initially, we planned on visualising the building in 3D, which would require a different library with a 3D rendering utilities, such as React Native 3D Model View package.

The app provides functionality for displaying points of interest, although there is currently no option for user to enter new points. Initially we planned on collecting additional data from users after they reach the final destination and automatically update points of interest on the map using the collected data.

Lastly, our solution was meant to be integrated with Open Street Map project. User can be unfamiliar with a building itself but also with the location of the building. Integration with Open Street Map would enable user to use already implemented outdoor navigation systems to find a target building (e.g. Sir William Henry Bragg building) and then the navigation would switch to our indoor navigation solution, so the user could find a room.

# Chapter 7

## Evaluation

### 7.1 Experiment

To evaluate the app, we designed an experiment to compare navigation provided by the app and traditional navigation methods (e.g. using signs and door plates). The experiment consisted of two groups: a baseline group that did not use the app, and a treatment group that did. Participants of each of these groups were given a survey which helped to evaluate the effectiveness of the methods they used and to gather demographic information.

#### Hypothesis

The use of the app aids participants in finding and travelling to rooms regardless of prior knowledge of the building.

The experiment consisted of gathering participants and asking them to find specific rooms. They were accompanied by two invigilators: one measuring times and one recording metrics, such as turns, stops and floor changes. The invigilators' responsibilities are explained in Appendix F.3.

#### 7.1.1 Sampling

We chose participants by convenience, offering an incentive of cookies. The majority of participants selected were currently within the Bragg building and were chosen for their willingness to complete the evaluation. For this reason, the data skewed towards people who were staff or students at the University of Leeds.

#### 7.1.2 Metrics

The metrics were selected to measure each participant's ability to navigate in the building and to make them comparable between the baseline and treatment groups. Initially, we wanted to measure time and distance travelled before reaching goal destinations. Unfortunately, distance proved itself to be infeasible to measure, so we took the number of turns, stops, and floor changes instead.

We started recording time as soon as the room name had been given to the participant and stopped once the participant touched the door of the given room. The timer was stopped whilst the participant was changing floors; this was to prevent the data from being affected by transition times between floors (be it via lifts or stairs).

We defined a turn to be rotation greater than a  $45^\circ$  angle, followed by movement of three steps in that direction. These turns by definition also account for U-turns ( $180^\circ$  turn). We did not count turns onto or off of a staircase or lift.

Stops were defined as any time the participant came to a complete stop for more than two seconds; this was likely to be because they were reading signs or were interacting with the app. More stops may indicate that the participant was trying to gather more information about their surroundings.

We measured the number of times participants changed floors. In some cases, participants changed floors more than they needed to. If a group had fewer floor changes, it might indicate they understood their position in the building better. We also recorded the ways people changed floors; whether they took the stairs or the lifts.

### 7.1.3 Search Locations

For consistency, we gave the same four room locations to every participant. We chose rooms on different floors to eliminate the chance that someone would know the location of all the rooms.

1. 24h-hour lab 2.15
2. Prof. Raymond's office 3.34
3. Kitchen/Print room 1.48
4. Gender neutral bathroom GR.26

The difficulty in finding each room in the experiment varies; the 24-hour lab and Prof. Raymond's office are relatively easy to find, whereas the other two rooms are harder. The signs pointing to the Kitchen/Print room (1.48) are incorrect (see Figure 1.1), and the gender-neutral bathroom (GR.26) is behind the café and has no signs pointing to it specifically.

### 7.1.4 Survey

Once the participants finished searching for the rooms, they were given a survey. This was to collect qualitative data from the experiment: demographic information about the participants, additional data related to the room searching, and general feedback. The survey questions can be found in Appendix F.6.

### 7.1.5 Baseline Experiment

The first of the two experiments was the baseline, focusing on the efficacy of travelling between two rooms in the building using existing signage. We did not allow participants to ask other people in the building for help, as that would not reflect the difference in using signs and the app. We also did not allow participants to use the large interactive map in the foyer as it was introduced after we started experiments and could hence potentially lead to inconsistent results.

The experiment started after the participant read the information sheet (Appendix F.2), instructions, and signed the consent form. Each participant was taken through the same procedure:

1. Give the participant the instructions in Appendix F.4 and get consent for the experiment (using Appendix F.1).
2. Give the participant the room to travel to.
3. Start the timer as soon as the room is given.
4. If a participant takes a turn or stops mark these in the notes.
5. When the participant touches the door of the room the invigilators were directed to stop the timer and mark down the time taken.
6. Repeat steps 2–5 for all the rooms in the test 7.1.3.
7. Once the list of rooms has been completed the participant is returned to the start to complete the survey and claim their incentive.

### 7.1.6 App Experiment

This second experiment required a participant to have access to the app. We decided to provide an Android phone with the app installed ourselves, to save time with additional setup on participants' phones.

The same steps outlined in the baseline were taken, differing only in that the instruction sheet in Appendix F.5 was given instead, and that the participants were prompted to use the navigation app.

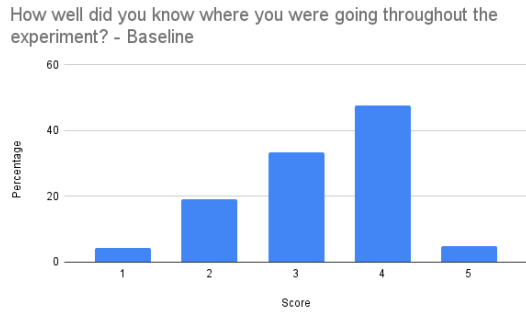
## 7.2 Results

### 7.2.1 Baseline Survey

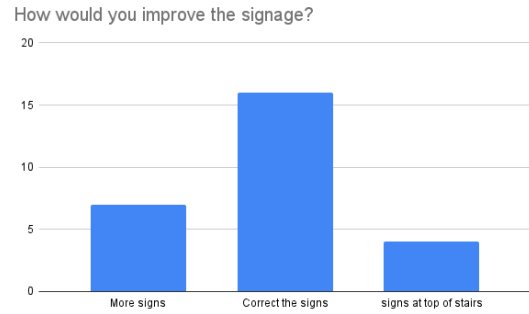
In the survey we included questions about the signage quality and invited participants to share potential improvements. Overall, the participants felt confident that they knew where they were going throughout the experiment. [Figure 7.1a]

**“How would you improve the signage?”** The most common response to this question was that the signs were wrong. The signage for the kitchen was incorrect; two of the signs for this room pointed in opposite directions to the kitchen. Other comments included the distance between consecutive door numbers (e.g GR.25 and GR.26 are on opposite sides of the building). [Figure 7.1b]

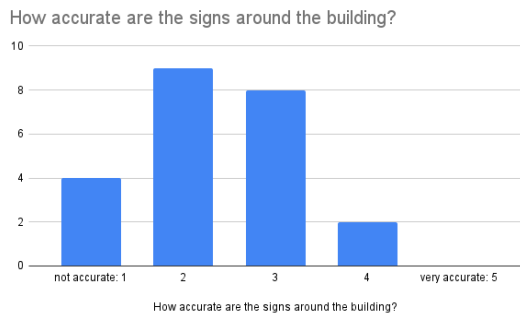
**“Is there anything that could help you with navigation in the building?”** Here we noticed three types of answers: the first pointed out inaccuracies in the signs (see Figure 7.1c), the second mentioned easily accessible maps, and the third talked about the creation of a



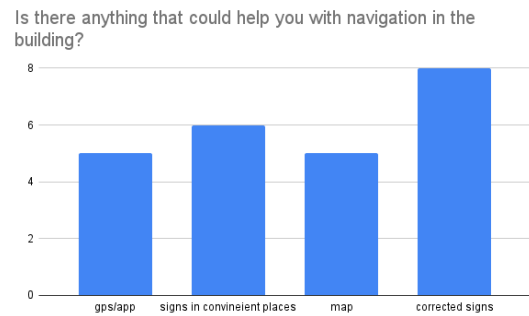
(a) Participant’s confidence as to how sure they were of where they were throughout the experiment



(b) How would participants improve signage



(c) How accurate the participants found the signs in the building



(d) Participant requests as to what would have aided them most in finding a location.

Figure 7.1: Questions from the Baseline Survey

system of localisation using an app in the building (see Figure 7.1d). In the final case it should be noted that we tried to not tell participants about our project to create such an app.

### 7.2.2 App Survey

During the experiment, it was found that the path suggested from room 2 to room 3 sometimes routed users back down to the first floor. This is due to an error in the path finding algorithm that doesn't weight lifts and staircases correctly.

We also received feedback that we should mark staircases in a different way to regular rooms. Since these have metadata attached that differentiate them already, we could easily change the colour or add an icon in the center.

Wi-Fi scans took five seconds to complete, meaning that the location data provided was never fresh. This caused some participants to be misled, even if the old data was accurate. This is strictly a hardware limitation and is discussed in Chapter 4.4.2.3.

Participants requested the functionality to rotate the map. This could be implemented in two ways; the first being a gesture, similar to Google Maps, allowing manual reorientation of the map using two fingers. Alternatively, sensor-based rotation could be used, which would align the map to the direction of the internal compass in the user's device.

A lot of users found it difficult to navigate as the performance of the app was slow, especially when room labels were enabled. This issue is further discussed in Section 6.2.2.

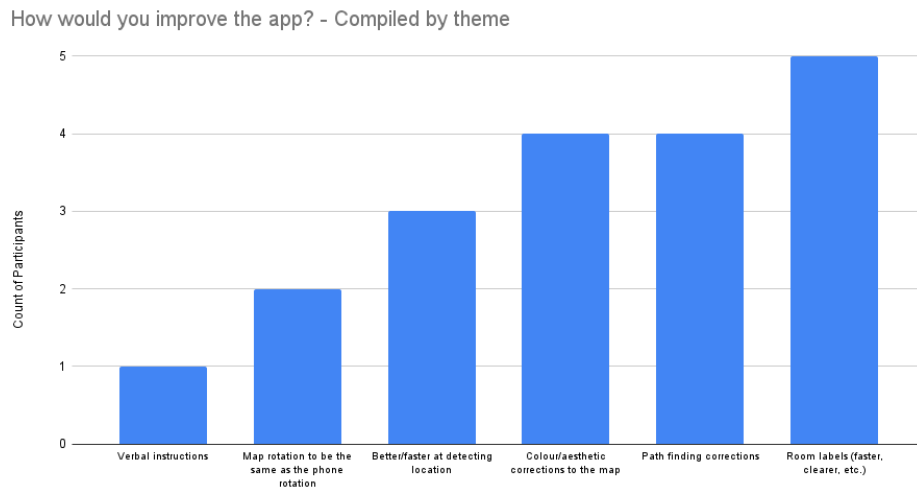


Figure 7.2: Themes compiled from the long-answer questions about improving the app



(a) How accurate was the app to predict your location?



(b) How easy was the path to follow?



(c) How aesthetically pleasing is the app?

Figure 7.3: Questions from the App Survey



### 7.2.3 Comparison of Results

The following headings refer to the graphs on the following page, comparing the metrics between the baseline experiment with the app experiment.

**Room 1 (24-hour visualisation lab)** Data for room 1 shows that the baseline was faster in both cases of familiarity and non-familiarity with the building. Both cases show a nearly two times increase in their average time (See Figure 7.5a). We believe this is due to users familiarising themselves with the app.

**Room 2 (Prof. Raymond's office 3.34)** We saw a small improvement in the time for room 2 in the familiar case, but the baseline was still faster in the non-familiar case. A potential explanation here could be over-confidence in the baseline experiment.

**Room 3 (Kitchen 1.48)** This room was the only room in the app experiment to include any dropouts.

The dropout in the app case was likely caused by an error in the map which routed users down two floors, then back up one, confusing them (see Section 7.2.2).

**Room 4 (Gender neutral bathroom GR.26)** This room was chosen to be difficult to find due to its signage. However, using the app's search users were able to locate it on the map. These results show the clearest benefit, both eliminating dropouts completely and resulting in a speedup in both cases.

This suggests that the app is good for finding obscure, hidden rooms, with limited signage.

**General Trends** In the case of the last two rooms, the stops and turns metrics were lowered significantly. This suggests that the users were generally less lost; these metrics are around the same for the first two rooms.

Floor changes remained reasonably constant between all tests, except the third. This implies that people probably knew the floor a room was on by its name on the card. In the case of the third room: in the baseline test some participants followed incorrect signs and went downstairs to the ground floor, and in the app test the bug outlined in Section 7.2.2 likely increased this metric.

**Hypothesis: The use of the app aids participants in finding and travelling to rooms regardless of past knowledge of the building**

The app aids participants that report they are not familiar with the building; in the case of those who say they are familiar, we saw less improvement.

The early rooms may have taken longer to find based on participants learning to use the app. This overhead might be something which would diminish over time given experience with the app; an issue that could be overcome by adding more rooms to the experiments.

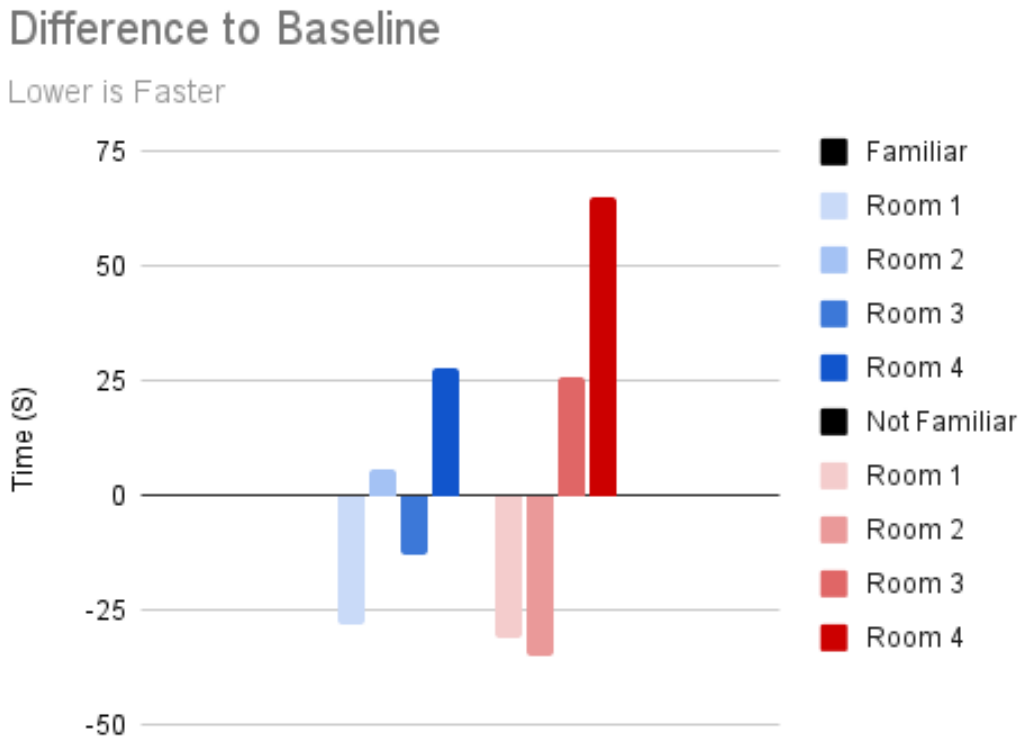


Figure 7.4: The difference in average times to baseline; negative values indicate the baseline test was faster

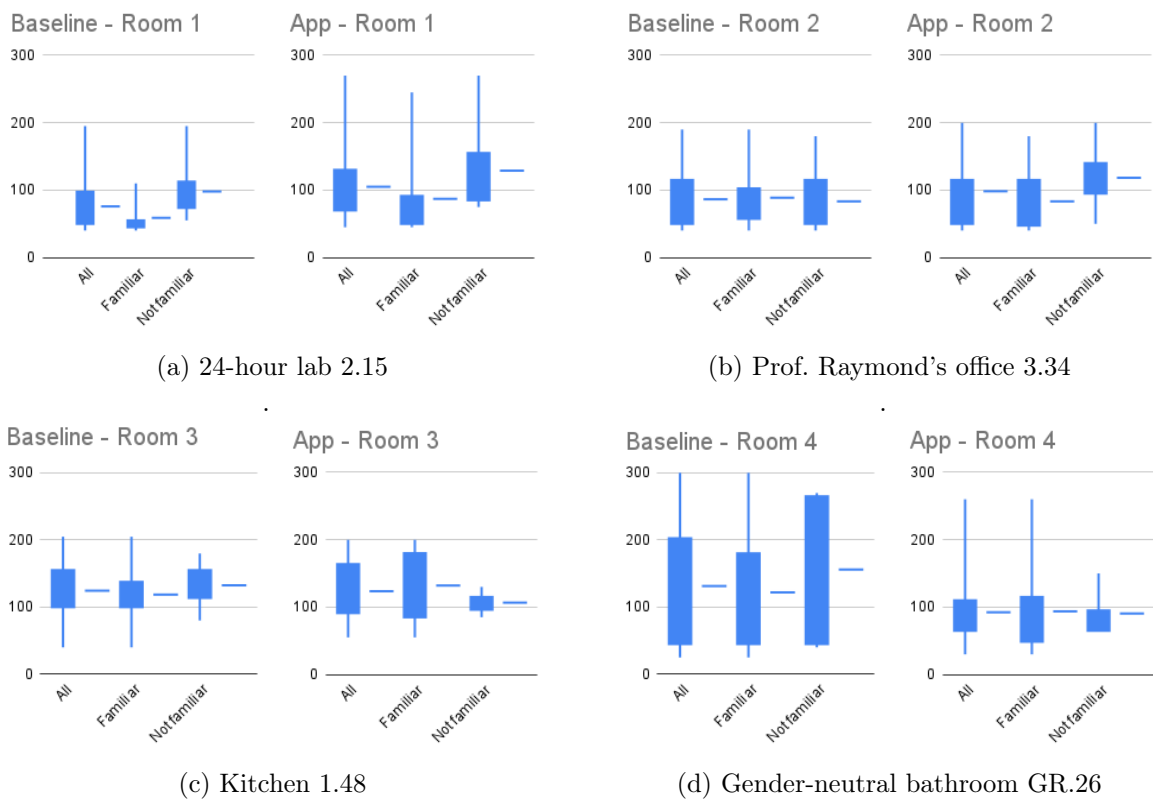


Figure 7.5: Times taken per room

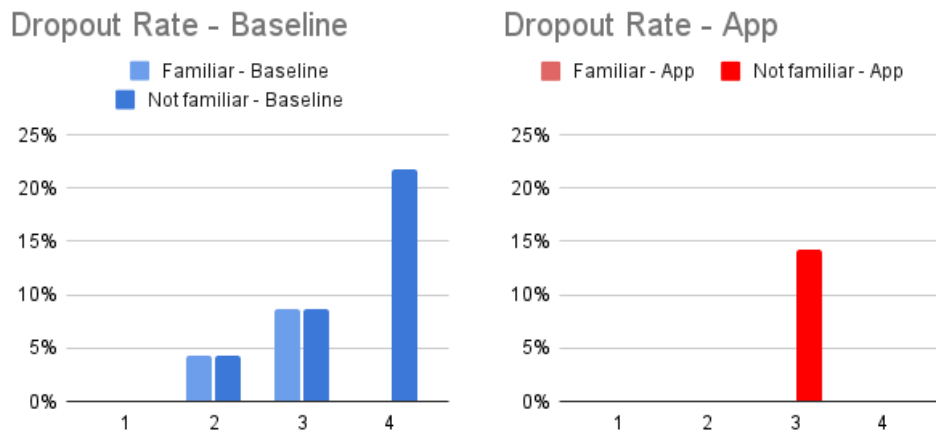
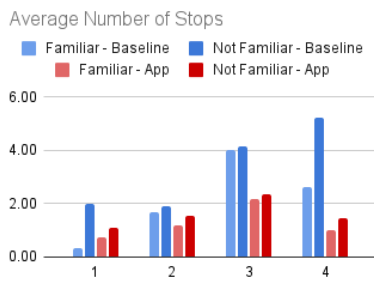
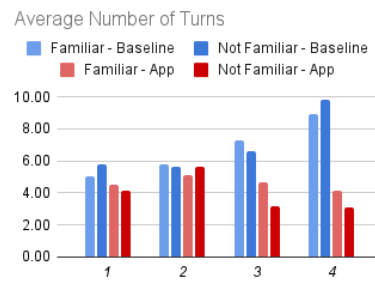


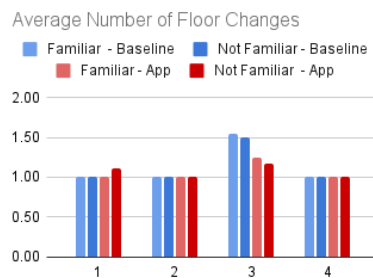
Figure 7.6: Dropout rate



(a) The average number of times participants stopped



(b) Average number of times participants made a turn



(c) Average floor changes participants made

Figure 7.7: Turns, Stops and Floor Changes.

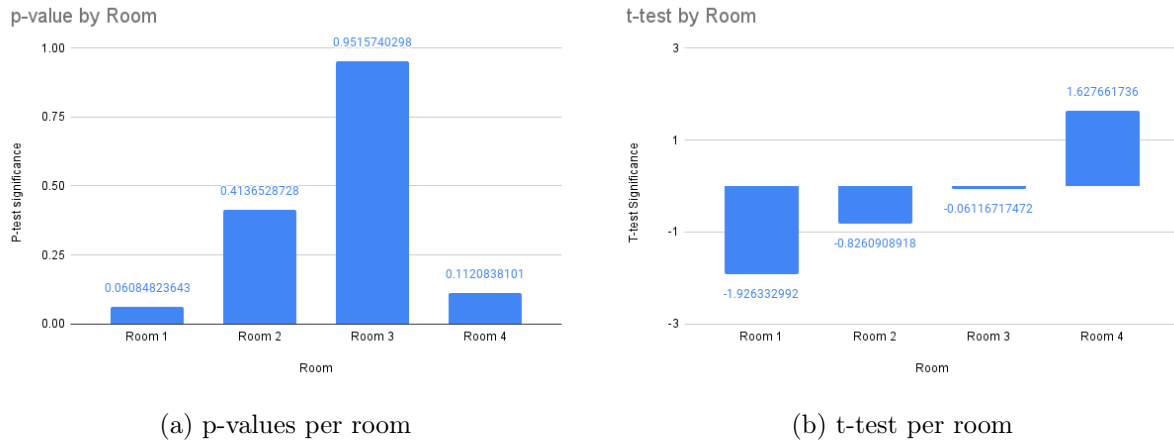


Figure 7.8: Statistical significance measures

In the app experiment, there may have been bias due to familiarity; participants may have not used the app and went straight to the first two rooms by memory.

We can see that our experiment had limited statistical significance (Figure 7.8a), so more testing should be completed to make any concrete claims.

### 7.2.4 Survey Comparison

#### How well did you know where you were going throughout the experiment?

Participants scored themselves 1–5 on if they knew where they were in the building: the average value was 3.26 in the baseline and 3.48 in the app experiment. This suggests that the app did help with the perception of localisation.

When asked participants were almost twice as likely to say the app’s given path was easy to use 57% with a score of 4–5 as opposed to the ease of use of the signs.

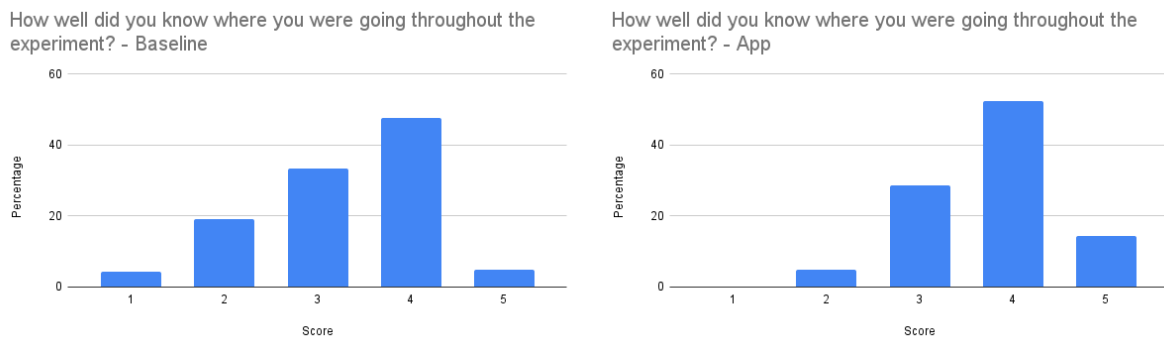


Figure 7.9: How confident were users in their location?

### 7.3 Revised Procedures

Following the first experiment and feedback from our supervisor, we decided that if we were to have had time to run this experiment again we would change how it operated.

More work should be done to define questions about familiarity with the building. We ask for familiarity with specific floors or rooms rather than about the building in general.

The more qualitative questions should have also been developed and tested to ensure they were interpreted consistently. This would mean we could make better conclusions from them; as they stand, they do not tell us much other than a general idea about our experiments.

During the app experiment participants were not restricted to navigating using only the app. Some participants barely used it; if we ran the app experiments again, we would specify that the participants should mainly use the app instead of the signs. This would give us a better comparison of searching rooms with and without the app.

The turns and stops were not optimal metrics to take. We thought that turns could reflect the distance travelled between rooms, but some paths of similar distance required participants to do a different number of turns, making the data unclear.

Increasing the number of participants would limit bias and result in more statistically significant data.

Using only one survey at the end of the experiment made it hard to control bias over some questions, such as familiarity. We could ask demographic questions at the start of the experiment, and experience-based questions at the end.

Another hypothesis to explore would be that the app helps users find obscure rooms more consistently. A focus on harder to find rooms may reveal further information.

# Chapter 8

## Conclusion

Overall, we have a good proof-of-concept that mapping and localisation using our methods is feasible, and that use of it may help in navigating commercial buildings such as Bragg. More work can be done to improve existing implementations, but our solution is sufficient for basic navigation.

### 8.1 Mapping

The scheme for mapping is open to extension but closed to modification and, whilst time-consuming, it is easy to create maps. The solution is hampered by software support for multiple floors, meaning there had to be implementation server-side. Overall the map created of the Bragg building was fit for the purpose of navigating the building.

### 8.2 Localisation

Localisation within the app performs well, proving sufficient for our use case. Limitations due to implementation details and hardware do exist; however, the app is able to follow the users position within a range of on average between 4 and 6 meters. This is appropriate for an indoor navigation system.

A parameter tuning algorithm was successfully implemented. This expanded on the scope of the background research performed, finding a combination of parameters outside of the ranges we predicted. These new parameters *felt* better when tested on the app, while showing a demonstrable improvement when assessed on 450 scans of input data.

Future work in this scope includes testing how the improved localisation method responds to data starvation, as well as the potential implementation of newer localisation methods such as Wi-Fi RTT. Closer integration with the rest of the app would also be appropriate, via methods such as path-snapping or weighting locations.

### 8.3 Implementation

The server implementation is good, and is suitable for our solution. It is flexible to changing mapping requirements, but not scalable. At the scale we are using it, it is fast: full-text search works quickly and is only limited by connection speed.

The app has shown promise in aiding navigation indoors for users who do not know a building. Further testing is needed to see if this app would help long-term for users of the building. Similarly, more testing would need to be done to see if the detrimental effect of using the app

in the first two rooms was caused by an unfamiliarity with its operation or from failings of the app.

## 8.4 Teamwork

As a team we worked well together, with all team members contributing in a significant way to the project. We were also able to adapt to changing situations and plans. Despite how well we worked as a team we was unable to stick to the plan we originally conceived in chapter 2. This was due to time constraints, scheduling conflicts and issues outside of our control. However, we were able to communicate effectively to understand people's situations and scale back unnecessary features.

# References

- [1] Android developers. Wi-fi location: ranging with rtt, 2022. <https://developer.android.com/guide/topics/connectivity/wifi-rtt#supported-devices> [Online; accessed 10-April-2022].
- [2] Android developers. Wi-fi scanning overview, 2022. <https://developer.android.com/guide/topics/connectivity/wifi-scan> [Online; accessed 19-April-2022].
- [3] Apollo GraphQL. Apollo graphql, 2022. <https://www.apollographql.com/>, [Online; accessed 22-April-2022].
- [4] Richard Atterer. Leadme - map creation for pedestrian navigation, 2010. <http://atterer.org/leadme>, [Online; accessed 4-april-2022].
- [5] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development, 2001. URL <http://www.agilemanifesto.org/>.
- [6] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1978. ISBN 0201006502.
- [7] brwhiz. Opry mills mall entrance 1, 2013. [https://www.waymarking.com/waymarks/WMHNMB\\_Opry\\_Mills\\_Mall\\_Entrance\\_1](https://www.waymarking.com/waymarks/WMHNMB_Opry_Mills_Mall_Entrance_1) [Online; accessed 25-April-2022].
- [8] Business Wire. Indoor retail mapping leader aisle411 delivers in-store 3d mapping on google’s project tango, 2014. <https://www.youtube.com/watch?v=6QHauAnecUM> [Online; accessed 4-April-2022].
- [9] Jorge Chen and Keith C. Clarke. Indoor cartography. *Cartography and Geographic Information Science*, 47(2):95–109, 2020. doi: 10.1080/15230406.2019.1619482. URL <https://doi.org/10.1080/15230406.2019.1619482>.
- [10] Dell. How to identify and reduce wireless signal innterference, 2021. <https://www.dell.com/support/kbdoc/en-uk/000150359/how-to-identify-and-reduce-wireless-signal-interference> [Online; accessed 20-April-2022].
- [11] Discord. Discord your place to talk and hang out, 2022. <https://discord.com> [Online; accessed 10-April-2022].



- [12] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006. doi: 10.1109/MRA.2006.1638022.
- [13] Erin Rodrigue. What is a qr code + how does it work? everything marketers should know, 2021. <https://blog.hubspot.com/blog/tabid/6307/bid/16088/everything-a-marketer-should-know-about-qr-codes.aspx> [Online; accessed 6-April-2022].
- [14] Amir Guidara, Ghofrane Fersi, Faouzi Derbel, and Maher Ben Jemaa. Impacts of temperature and humidity variations on rssi in indoor wireless sensor networks. *Procedia Computer Science*, 126:1072–1081, 2018. ISSN 1877-0509. doi: <https://doi.org/10.1016/j.procs.2018.08.044>. URL <https://www.sciencedirect.com/science/article/pii/S187705091831322X>. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 22nd International Conference, KES-2018, Belgrade, Serbia.
- [15] Janne Haverinen and Anssi Kemppainen. Global indoor self-localization based on the ambient magnetic field. *Robotics and Autonomous Systems*, 57(10):1028–1035, 2009. ISSN 0921-8890. doi: <https://doi.org/10.1016/j.robot.2009.07.018>. URL <https://www.sciencedirect.com/science/article/pii/S0921889009001092>. 5th International Conference on Computational Intelligence, Robotics and Autonomous Systems (5th CIRAS).
- [16] Huawei. The world’s first 5g indoor positioning — verified by china mobile suzhou and huawei, 2021. <https://www.huawei.com/en/news/2021/3/5g-indoor-positioning-china-mobile-suzhou>, [Online; accessed 22-April-2022].
- [17] Jim Katsandres. Bluetooth low energy – it starts with advertising, 2017. <https://www.bluetooth.com/blog/bluetooth-low-energy-it-starts-with-advertising/> [Online; accessed 6-April-2022].
- [18] Jorge Chen & Keith C. Clarke. Indoor cartography, 2018. <https://www.tandfonline.com/doi/full/10.1080/15230406.2019.1619482> [Online; accessed 7-April-2022].
- [19] Lucid Content Team. 4 phases of rapid application development methodology, 2022. <https://www.lucidchart.com/blog/rapid-application-development-methodology> [Online; accessed 18-April-2022].
- [20] MDN. Network information api - web apis | mdn, 2022. [https://developer.mozilla.org/en-US/docs/Web/API/Network\\_Information\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Network_Information_API) [Online; accessed 19-April-2022].
- [21] Official Journal of the European Union. General data protection regulation, 2016. <https://gdpr-info.eu/> [Online; Accessed 26-April-2022].
- [22] OGC. Indoorgml, 2019. <http://indoorgml.net/> [Online; accessed 4-April-2022].

- [23] OpenStreetMap Wiki. Way, 2021. <https://wiki.openstreetmap.org/w/index.php?title=Way&oldid=2173770> [Online; accessed 22-November-2021].
- [24] OpenStreetMap Wiki. Indoor mapping — openstreetmap wiki,, 2022. [https://wiki.openstreetmap.org/w/index.php?title=Indoor\\_Mapping&oldid=2285214](https://wiki.openstreetmap.org/w/index.php?title=Indoor_Mapping&oldid=2285214) [Online; accessed 4-April-2022].
- [25] Python Software Foundation. multiprocessing — Process-based parallelism — Python 3.10.4 documentation, 2022. <https://docs.python.org/3/library/multiprocessing.html> [Online; accessed 12-March-2022].
- [26] Python Wiki. GlobalInterpreterLock — Python Wiki, 2020. <https://wiki.python.org/moin/GlobalInterpreterLock> [Online; accessed 19-April-2022].
- [27] Scott - Sprout QR. How do qr codes work? qr code technical basics, 2020. <https://www.sproutqr.com/blog/how-do-qr-codes-work> [Online; accessed 6-April-2022].
- [28] Scrum.org. What is scrum?, 2022. <https://www.scrum.org/resources/what-is-scrum> [Online; accessed 20-April-2022].
- [29] Senion. How accurate are indoor positioning systems?, 2019. <https://senion.com/insights/accurate-indoor-positioning-systems/> [Online; accessed 7-April-2022].
- [30] StatCounter. Android version market share worldwide, 2022. <https://gs.statcounter.com/os-version-market-share/android> [Online; accessed 19-April-2022].
- [31] Junhua Yang, Yong Li, and Wei Cheng. An improved geometric algorithm for indoor localization. *International Journal of Distributed Sensor Networks*, 14:155014771876737, 03 2018. doi: 10.1177/1550147718767376.

# Appendices

# Appendix A

## Source Code

### A.1 Project Repository

[https://gitlab.com/comp5530m-mapping-project/comp5530m\\_mapping\\_project](https://gitlab.com/comp5530m-mapping-project/comp5530m_mapping_project)

### A.2 Map Repository

<https://gitlab.com/comp5530m-mapping-project/example-maps>

### A.3 Video demo

<https://www.youtube.com/watch?v=Wyd1FNRDyYw>

### A.4 Survey

<https://forms.gle/fBCQBnDYbWsmxUaT7>

### A.5 Consent Forms

The scanned consent forms are available on request.

### A.6 Experiment Data & Survey Results

These are available on request.

# Appendix B

## List of Libraries / Software Used

### B.1 Server

#### B.1.1 Software

- `gdal` — Conversion between geographic data formats
- `Redis` — Database
- `Docker` — Development containers, deploy
- `docker-compose` — Container composition, deploy
- `Pants` — Build system
- `flake8` — Linter
- `pytest` — Tests
- `Azure App Service` — Cloud platform for container deployment
- `Jupyter` — Notebook for prototyping

#### B.1.2 Python libraries

- `networkx` — Path finding, graph algorithms
- `redis` — Database
- `redisgraph` — Graph database
- `redisearch` — Full-text search
- `ariadne` — GraphQL API definition
- `pydantic` — Dataclass validation
- `shapely` — Computational geometry
- `uvicorn` — Serving GraphQL API
- `pyproj` — Coordinate projection

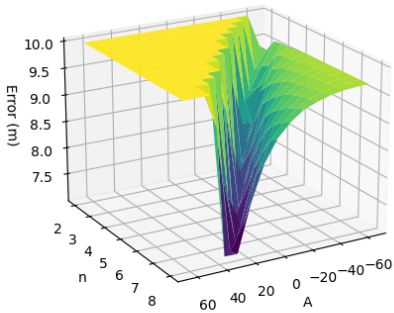
### B.2 Client

Important packages from `app/package.json`:

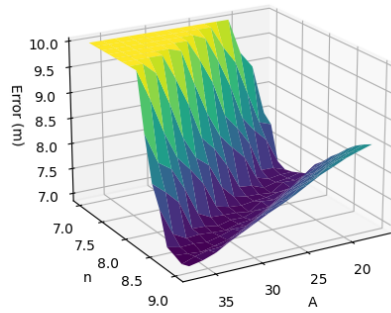
- `@apollo/client` — GraphQL client for querying server
- `@react-navigation/native` — Implements native navigation for apps
- `@react-navigation/stack` — Stack navigator for React Navigation package
- `d3` — Data handling and transformation library
- `expo-sensors` — For gyrometer readings
- `geodesy` — Handles geodesic information
- `react` — Base React framework
- `react-native` — Allows native development for Android and iOS
- `react-native-compass-heading` — Handles compass readings
- `react-native-plotly` — Plot library
- `react-native-svg-pan-zoom` — SVG rendering with gesture support
- `react-native-wifi-reborn` — Handles Wi-Fi information and scanning

# Appendix C

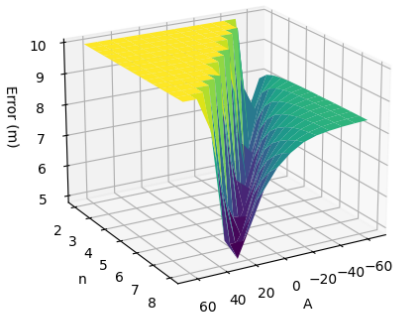
## Parameter Tuning Graphs



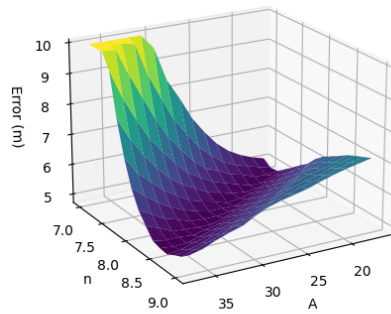
(a) A: 64 to -64dBm | n: 2 to 8  
Wide scan over the ground floor.



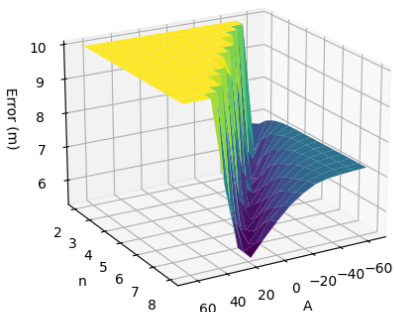
(b) A: 37 to 17dBm | n: 7 to 9  
Narrow scan over the ground floor.



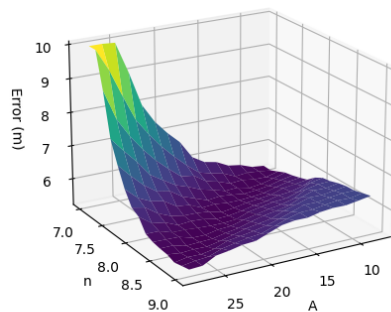
(c) A: 64 to -64dBm | n: 2 to 8  
Wide scan over floor 1.



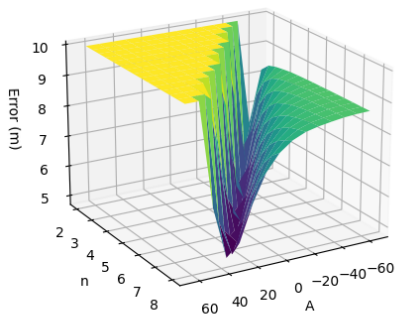
(d) A: 37 to 17dBm | n: 7 to 9  
Narrow scan over floor 1.



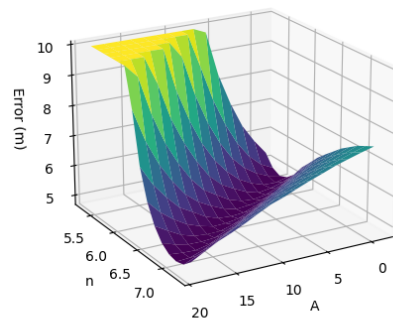
(e) A: 64 to -64dBm | n: 2 to 8  
Wide scan over floor 2.



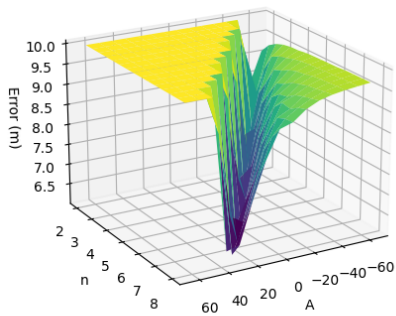
(f) A: 28 to 8dBm | n: 7 to 9  
Narrow scan over floor 2.



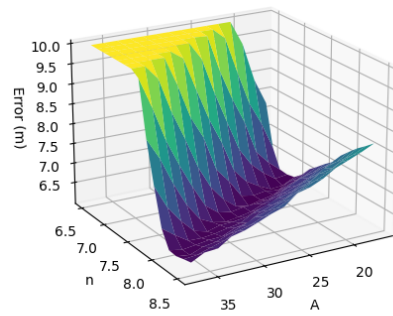
(g) A: 64 to -64dBm | n: 2 to 8  
Wide scan over floor 3.



(h) A: 19 to -1dBm | n: 5.3 to 7.3  
Narrow scan over floor 3.



(i) A: 64 to -64dBm | n: 2 to 8  
Wide scan over floor 4.



(j) A: 37 to 17dBm | n: 6.5 to 8.5  
Narrow scan over floor 4.

Figure C.1: Graphs of average errors (n) in parameter tuning process. A lower result is better.



# Appendix D

## Parameter Tuning Tables

Point	Floor					Average
	Ground	1	2	3	4	
1	3.24	6.84	3.94	3.89	14.16	6.41
2	8.89	6.95	3.54	11.72	4.14	7.05
3	5.97	7.87	3.14	11.26	15.64	8.78
4	4.58	6.54	10.57	7.89	4.49	6.81
5	11.99	9.88	2.75	2.69	6.20	6.70
6	12.99	5.10	4.30	7.51	6.35	7.25
7	10.16	9.50	10.11	7.66	—	9.36
8	17.26	4.90	9.09	13.87	—	11.28
9	6.08	10.75	7.56	3.11	—	6.88
10	9.02	4.13	6.11	—	—	6.42
Average	9.02	7.25	6.11	7.73	8.50	<b>7.72</b>
Std. Dev.	4.06	2.13	2.86	3.77	4.62	<b>1.52</b>

Figure D.1: The error (m) at each point on each floor with the base parameters.  
A: -50dBm | n: 3

Point	Floor					Average
	Ground	1	2	3	4	
1	2.08	6.58	2.35	2.27	8.31	4.32
2	7.08	2.73	2.50	7.39	5.44	5.03
3	4.99	5.20	1.80	6.85	11.64	6.10
4	3.05	3.48	8.03	5.98	3.09	4.73
5	5.18	7.80	3.39	3.93	3.74	4.81
6	9.37	1.29	3.77	4.68	5.12	4.85
7	6.00	5.24	15.20	1.48	—	6.98
8	22.96	2.28	9.40	10.89	—	11.38
9	4.10	6.35	9.33	2.01	—	5.45
10	5.51	7.28	3.75	—	—	5.51
Average	7.03	4.82	5.95	5.05	6.22	<b>5.91</b>
Std. Dev.	5.65	2.14	4.14	2.88	2.93	<b>1.96</b>

Figure D.2: The error (m) at each point on each floor with the new parameters.  
A: 25.7dBm | n: 7.71

# Appendix E

## Listings

Listing E.1: GraphQL Schema

```
type Query {
  node(graph: String!, id: Int!): Node
  nodes(graph: String!): [Node!]
  search_nodes(graph: String!, search: String!): [Node!]
  walls(graph: String!): [Node!]

  edges(graph: String!): [Edge!]

  poi(graph: String!, id: Int!): PoI
  pois(graph: String!): [PoI!]

  search_pois(search: String!): [PoI!]
  search_pois_in_graph(graph: String!, search: String!): [PoI!]

  polygon(graph: String!, id: Int!): Polygon
  polygons(graph: String!): [Polygon!]
  search_polygons(graph: String!, search: String!): [Polygon!]

  find_route(graph: String!, start_id: Int!, end_id: Int!): Path!
}

type Mutation {
  add_graph(graph: String!, polygons: String!, points: String!, linestring:
    String!): Boolean!
  flush_all: Boolean!
}

type Node {
  id: Int!
  graph: String!
  level: Float!
  lat: Float!
  lon: Float!
  polygon: Polygon
  tags: Tags
  neighbours: [Node!]
}

type Polygon {
  id: Int!
  graph: String!
  level: Float!
  sw: [Float!]!
  ne: [Float!]!
```

```
    vertices: [[Float!]]!  
    tags: Tags  
}  
  
type Poi {  
    id: Int!  
    graph: String!  
    level: Float!  
    lat: Float!  
    lon: Float!  
    tags: Tags  
    nearest_path_node: Node  
}  
  
type Path {  
    ids: [Int!]!  
    nodes: [Node!]!  
    instructions: [String!]!  
    levels: [Float!]!  
}  
  
type Edge {  
    edge: [Int!]!  
    graph: String!  
    adjacent_nodes: [Node!]!  
}  
  
scalar Tags
```

# Appendix F

## Evaluation

### F.1 Consent Form

Consent to take part in Indoor Mapping Project Add your initials next to the statement if you agree

	Initial this box to indicate consent
I understand that my name will not be linked with the research materials, and I will not be identified or identifiable in the report or reports that result from the research. I understand that my responses will be kept strictly confidential	
I confirm that I have read and understand the information sheet explaining the above research project and I have had the opportunity to ask questions about the project	
I understand that my participation is voluntary and that I am free to withdraw at any time without giving any reason and without there being any negative consequences. In addition, should I not wish to answer any particular question or questions, I am free to decline. Until the end of day of the experiment and before data can be compiled completely the data will be removed from the pool and no further data will be collected.	
I understand that members of the research team may have access to my anonymised responses.	
I understand that my name will not be linked with the research materials, and I will not be identified or identifiable in the report or reports that result from the research. I understand that my responses will be kept strictly confidential	
I understand that the data collected from me may be stored and used in relevant future research in an anonymised form	
I understand that relevant sections of the data collected during the study, may be looked at by individuals from the University of Leeds or from regulatory authorities where it is relevant to my taking part in this research.	
I agree to take part in the above research project and will inform the lead researcher should my contact details change.	

Name of participant's signature	
Date	
Name of lead researcher [or person taking consent]	
Signature	
Date*	

\*To be signed and dated in the presence of the participant.

## F.2 Information Sheet

**Title of the Project: Indoor Mapping** You are being invited to take part in a research project. Before you decide it is important for you to understand why the research is being done and what it will involve. Please take time to read the following information carefully and discuss it with others if you wish. Ask us if there is anything that is not clear or if you would like more information. Take time to decide whether you wish to take part.

**Purpose of the project:** This project intends to create a functioning method of mapping and navigation for indoor areas and compare it against traditional means such as signposting in the aim of improving navigation methods for areas where GPS would be unsuitable.

**Do I have to take part?** It is up to you to decide whether to take part. If you do decide to take part, you will be given this information sheet to keep (and be asked to sign a consent form) and you can still withdraw at any time without it affecting any benefits that you are entitled to in any way. You do not have to give a reason.

**What do I have to do?** You will need to arrive at the William Bragg building for your assigned/chosen time slot and navigate between several indoor locations followed by a short survey. In all the experiment should take no longer than 1 hour (though very likely less) and at the end you will receive a slice of pizza. Travel expenses are not available for participation and should require no changes to lifestyle before or after the experiment is completed. The survey should last no longer than 5 minutes, though it is not timed, and will not require any personal information. The survey will be a mix of 1–5 scale questions, pick from a list of options questions, with some open questions which you are free to write as you like.

It is expected that on the day you will not ask directions from anyone in the building and will simply use the signs posted around the building to find the given locations. Then to finally go to the final room for the survey. It is asked that you be as honest as possible during the survey for the best results.

**What are the possible disadvantages and risks of taking part?** There are no known disadvantages or risks further than those that would be expected when travelling within a building.

**What are the possible benefits of taking part?** For starters, you will gain a slice of pizza given upon completion or end of the experiment as well as the team's gratitude.

**Dissemination and storage of research data** Any data gathered from this experiment will be stored securely with a password as well as data from the experiment not requiring any identifying data (other than on consent forms) with this data to only be used for the completion of these masters level module.

**What will happen to my personal information?** Your personal information on these consent forms will be held for no longer then is required by the university upon completion of the experiment and once this time elapses all personal information will be destroyed. However, the data gathered during the experiment will not include identifiable information and at most might have a number identifier given which will in no way be identifiable to you. The data will be compiled with other participants to form conclusions on the suitability of the available signage.

All the contact information that we collect about you during the course of the research will be kept strictly confidential and will be stored separately from the research data. We will take steps wherever possible to anonymise the research data so that you will not be identified in any reports or publications.

**What type of information will be sought from me and why is the collection of this information relevant for achieving the research project's objectives?** For the purpose of this experiment, we will take several measurements to define how easy it is for general travel between two points within the building. This information will later be used to contrast against an app of our creation in determining the usefulness of the signs and evaluate our performance.

**Who is organising/ funding the research?** This research is taken as a part of general studies in the university through the module COMP5530M Group Project and has gained no further/external funding.

Those in the group are:

- Supervisor: Evangelos Pournaras
- Kane Easby
- Usama Usman
- Matthew Pawson
- Thomas Carroll
- Tomas Martinek
- Samuel Palabiyik
- Kevan Jordan

For further information or to ask any questions please contact Samuel (Sam) Palabiyik at the email [sc18stp@leeds.ac.uk](mailto:sc18stp@leeds.ac.uk)

Or Supervisor Evangelos Pournaras at: [E.Pournaras@leeds.ac.uk](mailto:E.Pournaras@leeds.ac.uk)

You will be given a copy of the information sheet and, if appropriate, a signed consent form to keep.

Finally, from all of us in this project we would like to thank you for considering participation into this project and hope you will join us and get the easiest slice of pizza of your life

## F.3 Invigilator Instructions

### Rules

- Do not give the participants any further directions to the next room
- Do not allow them to stop for extended periods to talk or ask directions, start with a request, then if they ask directions end the experiment immediately, if it was simply a chat warn them of the possibility of early termination.
- Be sure to get valid consent before starting
- Have your stopwatch/notepad ready before giving the location
- Do not give out or allow them to see this document

### Instructions

1. Take the valid consent form and check all is filled in.
2. Sign and date where it calls for the invigilator in the consent form.
3. Check they have fully understood all instructions.
4. Give them the first room from the list
5. As soon as you give the room start the timer
6. Time and record the metrics, if the participant takes more than five minutes, mark as DNF and move to the next room
7. Follow them until they find the location given repeating step 6 as necessary
8. Repeat for All other rooms (noting the rooms in the Room Order which you are to lead them to)
9. Once the final room from the “Room Order” is found take them somewhere quiet for the survey and incentive.

## Room Order

1. Front entrance with revolving doors (to be lead there by the invigilator as close to the door as is reasonable)
2. 2.15 — 24h lab
3. 3.35 — Raymond's Kwan Office
4. 1.48 — Kitchen / Print
5. GR.26 — Gender neutral bathroom (behind café)
6. Incentive room — Depends on availability

## F.4 Participant Instructions — Baseline

We would like to thank you for your participation.

For the purpose of this document: Participant refers to you or the person volunteering for the experiment Invigilator refers to the person in the group who will join you through the experiment and give you the destination.

### Rules

- Once the experiment begins, we ask you not to speak to anyone other than the invigilator, including lecturers, other students, janitorial staff, etc.
- Please try to follow the signs/signposts around the building as best you can to find the locations.
- No help in finding locations will be provided by the invigilator\*.
- Please try to get to the destination as efficiently and quickly as possible while following the previous rules.

\*If you need reminding of the destination or room you are travelling to, feel free to ask, and it will be given to you.

### Instructions

1. The experiment will begin when you are told the location of the first room which we ask you to navigate to.
2. Once you are told the destination please walk to the destination. Hint: It is likely you will not know where the room is. Please try to find signs or signposts around the building to get information.
3. While travelling feel free to make any comments about the experiment, navigation or signs in the building known to the invigilator.
4. Move to the room you have been told, then come to a full stop in front of the door.



5. You will then be told the next location to travel to a new location, so please repeat what you did in steps 2–4.
6. Once the invigilator says the final room is found please follow them to the final room
7. You will now be asked to fill in a simple survey about the experience.
8. On completion, please take the reward from an invigilator.

## F.5 Participant instructions — App

We would like to thank you for your participation.

For the purpose of this document

- Participant refers to you or the person volunteering for the experiment.
- Invigilator refers to the person in the group who will join you through the experiment and give you the destination.

### Rules

- Once the experiment begins, we ask you not to speak to anyone other than the invigilator; including lecturers, other students, Janitorial staff, etc. \*
- Please try to follow the app around the building as best you can to find the locations as best as you can. Though you can still use signs if you need to.
- No help in finding locations will be provided by the invigilator.
- Please try to get to the destination as efficiently and quickly as possible while following the previous rules.
- Before you begin please read the App Instructions given.

\*If you need reminding of the destination or room you are travelling to, feel free to ask, and it will be given to you.

### Instructions

1. The experiment will begin when you are told the location of the first room which we ask you to navigate to.
2. Once you are told the destination please walk to the destination. Hint: It is likely you will not know where the room is. Please try to use the app where possible.
3. While travelling feel free to make any comments about the experiment, navigation or signs in the building known to the invigilator.
4. Move to the room you have been told, then come to a full stop in front of the door.

5. You will then be told the next location to travel to a new location, so please repeat what you did in steps 2–4.
6. Once the invigilator says the final room is found please follow them to the final room
7. You will now be asked to fill in a simple survey about the experience.
8. On completion, please take the reward from an invigilator.

## F.6 Survey Questions

### Demographic questions

- Are you familiar with the Sir William Henry Bragg Building? — Yes/No
- If ‘Yes’, how familiar are you with the building? — Likert scale
- Are you part of the School of Computing? — Yes/No
- Are you part of the University of Leeds? — Yes/No
- How would you rate the below methods for finding a room in a building?
  - Follow signs — Likert scale
  - Ask someone — Likert scale
  - Blindly searching — Likert scale
- What do you usually remember rooms by?
  - Location — Likert scale
  - Room number (e.g. 2.14) — Likert scale
  - Room name (official name, e.g. ‘Comp office’) — Likert scale
  - Room use (unofficial name, e.g. ‘Steve’s office’) — Likert scale
- What floor do you usually call the floor with the main entrance? — Ground Floor, First Floor, Other
- What kind of phone do you own? Android, iOS, other
- Is there anything that makes traversal of the building difficult for you? — Long answer
- Is there anything that could help you with navigation in the building? — Long answer
- How well did you understand the instructions given at the start of the experiment? — Likert scale

### Baseline Experiment

- If an alternate method of navigation was available, would you use it? — Yes/No

- How easy to use are the signs around the building? — Likert scale
- How accurate are the signs around the building? — Likert scale
- How well did you know where you were going throughout the experiment? — Likert scale
- How would you improve the signage? — Long answer

### **App Experiment**

- Would you use the app if it was publicly available? — Yes / No
- How easy was the app to use? — Likert scale
- How aesthetically pleasing was the app? — Likert scale
- How easy was the path given by the app to follow? — Likert scale
- How accurately did the app predict your location? — Likert scale
- How well did you know where you were going throughout the experiment? — Likert scale
- How would you improve the app? — Long answer

The final survey is as follows: <https://forms.gle/fBCQBnDYbWsmxUaT7>