NTNU
Norwegian University of
Science and Technology

Visualization of Crowd Flows from Positioning Data

Hans-Kristian Seem Koren

Master of Science in Computer Science
Submission date: June 2017
Supervisor: John Krogstie, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Positioning systems, both indoor and outdoor, regardless of technology used, generate large amounts of data of varying quality about where different objects are observed in space at various time instants. Such data has been proven valuable for both commercial interests and for researchers alike. Given the nature of positioning data, it is invaluable to have tools for visualizing them, with maps being a natural part of the visualizations.

Various efforts have been made to generate different visualization tools for data collected at the campus of the Norwegian University of Science and Technology, but they have only made use of small subsets of the available data, partly due to how it was stored. Later, work has been done on improving the storage mechanisms, suggesting that it should be combined with the other work.

This project investigates building a web application and a data processing component with the goal of providing a research platform for exploring a larger part of the data set that for example can aid in the development of data cleaning methods. The result is a set of patterns discovered using the platform and an evaluation of the efficency of using the web platform as a basis for large-scale visualization applications. Experiments show that there are certain limits to what is possible using current technologies, but that there still is a great possibility of visualizing large datasets.

Sammendrag

Både innendørs og utendørs posisjoneringssystemer, uavhengig av hvilken teknologi som benyttes, genererer en stor mengde data av varierende kvalitet om hvor ulike objekter observeres i rommet på ulike tidspunkt. Denne dataen har vist seg nyttig for både kommersielle interessenter og forskere. På grunn av dataens natur, er det uvurdelig med visualseringer, der kart fremstår som et naturlig utgangspunkt.

Ulike arbeider har allerede blitt gjort når det kommer til slike verktøy for data innsamlet på NTNU (Norges teknisk-naturvitenskapelige universitet), men bare en liten del av den tilgjengelige dataen har blitt benyttet, delvis grunnet hvordan den var lagret. Senere arbeider har sett på ulike databaseløsninger, som det har vært ønskelig å integrere.

Dette prosjektet undersøker hvordan en webapplikasjon med tilhørende bakenligende systemer kan lages med mål om å lage en forskningsplattform som kan støtte undersøkelser av datasettet, for eksemepl som hjelp til å utvikle datarensingsmetoder. Resultatet fra prosjektet er et sett med visualiseringer fra det utviklede programmet, og en evaluering av hvordan de benyttede teknologiene er egnet. Ekseperimenter viser at det finnes grenser for å hva som er praktisk mulig, men at en kan få utrettet mye.

Problem Description

Since 2009, MazeMap has been working on indoor navigation with our app. An area we want to investigate further is how we can use collected positioning data to aid building managers and other stakeholders. This involves processing huge amounts of data in a near real-time manner to produce anonymous statistics that have easily available real-world applications. Such a system could e.g. be used by shopping mall owners to base real estate leasing prices on the number of people visiting areas around a store. The task is to process position data to give a simple overview of crowd movements in a building, in the first place based on an existing dataset on data from the NTNU campus The task is to be done according to a design science research model, and the report is expected to be written in English. The application should be made available under an open source license if not otherwise decided

Preface

This thesis concludes my Master of Science and is the result of research conducted in my final semesters at the Department of Computer and Information Science (IDI) at Norwegian University of Science and Technology (NTNU).

I would like to thank Professor John Krogstie at the Department of Computer and Information Science (IDI), who have been my supervisor, and Dirk Ahlers for providing helpful feedback and advice throughout the project.

Hans-Kristian Seem Koren
June 2017

Contents

List of Tables

List of Figures

Glossary

AsterixDB A a scalable, open source Big Data Management System (BDMS). 6

DOM The Document Object Model. 45

GeoJSON A JSON based format for encoding a variety of geographic data structures.. 16

JavaScript A general purpose programming language. 12, 15, 18

MBTiles A file format for storing large amounts of tile data. 17

PostgreSQL The world's most advanced open source database (according to their description). 17

Acronyms

CPU Central Processing Unit. 18

CSS Cascading Style Sheets. 33

GIS Geographic Information System. 13

GPS Global Positioning System. 9

GPU Graphics Processing Unit. 12, 18, 73

IDI Department of Computer and Information Science. vii

IS Information Systems. 19

MSE Mobility Service Engine. 10

NPM Node Package Manager. 34

NTNU Norwegian University of Science and Technology. 1, 6, 55

PNG Portable Network Graphics. 14

SQL Structured Query Language. 17, 44, 56, 75

Chapter 1

Introduction

This chapter will provide a short introduction to the project and why the research was conducted. It begins with a section explaining the background and motivation for the project, followed by a description of the specific problem and outline for the rest of the thesis.

1.1 Background and Motivation

In recent years, there has been an increased interest in gathering positioning data from various venues. Such data are believed to provide great value to both businesses and venue owners, and has therefore been found as an interesting area of research. Examples of venues of interest include, but is not limited to universities, hospitals, airports and shopping malls. At Norwegian University of Science and Technology (NTNU) in Trondheim, there has been installed equipment to track users via WLAN access points throughout the entire campus and the recorded data have been stored for further analysis. More than 1,800 WLAN access points spread over 350,000 square meters university area[12] contributes enough data to perform interesting analysis on the whereabouts of people in this location. Prior work has been conducted by previous students in the same area. Some applications for exploring the datasets through visualization have been developed, but the amount of available data were limited. This project will extend upon their work and provide a system with a larger data foundation incorporating research on database solutions for storing the data. The data were automatically collected by third-party systems as single records pinpointed to a place at a given time. Thus, it is not known beforehand what large amounts of such data looks like when visualized and what kind of patterns that can be observed. For this reason, it is valuable to have a tool that can help digesting and make something out of the data.

1.2 Project Definition

This project has investigated if it is feasible to implement a web-based application that can handle large amounts of location tracking data and visualize aspects of this data such as snapshots at given time instants or time ranges, movement trajectories over user-defined periods of time and movement between objects of interest, particularly buildings. In addition, the application has been used to identify and reveal patterns in the available data that both can be actual patterns or be caused by external factors. The work has been done in connection to Wireless Trondheim Living Lab[4].

A short literature review of the field of study to gain insight into the problem domain, and an assessment of available tools were conducted to provide a better foundation. The project is based on earlier work by other students at NTNU, so some of the problems this project has tried to solve was to some degree already identified. The project followed a design science research approach discussed in Chapter 3.

Resulting from the project is a web-based application capable of providing visualizations of the available dataset through a user-friendly interface. The application was evaluated in terms of performance and what insight it could give to a researcher. As the application is currently tailored to researchers, it was not found useful to perform a thorough evaluation of the all the user-friendliness aspects.

1.3 Project Description and Contributions

The main contributions of this thesis are:

- Exploration of techniques for rendering large amounts of spatio-temporal data in a browser.
- An implementation of a web-based visualization tool for location tracking data including a server that transfers data from the database to the clients.

- Evaluation of such a system focusing on the limits of how much data it can handle and how it performs on typical personal computers with commodity hardware.
- A discussion of patterns revealed, what could cause them and how likely they are to be actual.

1.4 Outline

Following this introductory chapter, the thesis is structured as:

- **Chapter 2: Background Theory** This chapter introduces some theoretical background for the project and reviews a few concepts used in the project. It also serve to describe the problem awareness part of design science research.
- **Chapter 3: Research Methodology** A presentation of the research methodology used in this project.
- **Chapter 4: Problem Elaboration and Requirements** A thorough elaboration of the problem domain and the requirements for the application.
- **Chapter 5: The Application** This chapter presents the developed application in detail.
- **Chapter 6: Results** This chapter presents the results from the research.
- **Chapter 7: Discussion** This chapter discusses the results and evaluation of the artifact.
- **Chapter 8: Conclusion and Future Work** The final chapter concludes the thesis, presents the knowledge contribution and a list of suggested future work.

Chapter 2

Background Theory

This chapter will provide insight into the necessary background information to give a better understanding of prior research and the theoretical foundation upon which the solution is built. This includes a recap of related work at NTNU along with a literature study of topics touched in this research.

2.1 Related Work

In a previous work conducted by Aulie [10] in his master thesis, a system for visualizing both crowd distributions and movement based on data gathered from on-campus WLAN access points. The system presented different views of the positioning data, both static device locations, single user movements around campus and how accurate the data is using a web UI where the user could adjust the parameters. However, the system was limited as it only had a small subset of the total data set available, due to the lack of a proper data store. All processing and retrieval were done from a single large JSON file. The thesis also presents some algorithms for detecting single-user movements.

As part of the evaluation different stakeholders were interviewed, and the general consensus was that such applications were perceived as useful. Some of the feedback from the interviewees pointed out interesting interpretations of the data:

- Determining which entrances and exits are being used

- Knowing which areas were heavily crowded at particular times to determine when to perform maintenance and cleaning
- Find out where and from which directions people entered the campus
- Automatic counting of people in a room

The results from this research showed that what one can call trivial patterns such that traffic is larger in the middle of work days than in the weekends confirmed. No conclusions could be drawn on non-trivial patterns due to the data set constraints and the processing methods utilized. The researcher suggested future work on improving the data availability to improve the basis upon which conclusion could be drawn.

Eriksen's master thesis[18] also revolved around exploring the positioning data collected from the NTNU campus. His focus was mainly how such visualizations could aid facility managers in their daily work. The artifact developed was an application that could generate a heat map of crowd distributions on campus. Like Aulie, only a limited set of the data provided the foundation for visualization. Interviews with facility managers shed light on some things the data could be used for and areas of application:

- Replacement for manually counting people in a room
- Identify how an area is used in practice
- Emergency situations such as **Fire and evacuation** and **property management** to schedule cleaning and maintenance.
- Gain knowledge about where people are, and where they are *not*, to improve utilization.

In a later work, Kongshem [28] looked into how the position data could be stored more efficiently using different database systems for the benefit of visualization applications to be able to improve the prototypes, as suggested by the other researchers working on the same. He compared the three database solutions MySQL, MongoDB and AsterixDB examined them and proposed some queries that could possibly be answered. The databases examined can be classified as a relational database (RDBMS), NoSQL Document Store and a Big Data Management System (BDMS) respectively. The results showed that there are pros and cons with each of the solutions depending on which trade-offs being acceptable, and that none of them presents itself as the best solution in all possible scenarios. Some databases are very good at storing data, but fail to make retrieval queries fast. Others are good at reading, but worse at writing. Scalability differs between the system, and some can horizontally scale over a cluster of machines, while others are limited to a single machine but scale reasonably well vertically.

Holten [23] investigated if the dataset could be used to automatically count the number of people in different rooms by comparing to manual counts. The results showed that using it for this task with the current dataset was not really viable due to inaccuracies in the data rendering it difficult to exactly pinpoint people to specific rooms.

2.2 The Need for Visualization

McCormick et al.[11] describe visualization in these words:

> "Visualization is a method of computing. It transforms the symbolic into the geometric, enabling researchers to observe their simulations and computations. Visualization offers a method for seeing the unseen. It enriches the process of scientific discovery and fosters profound and unexpected insights. In many fields it is already revolutionizing the way scientists do science."

Visualizations are important for human interpretation of data. Computers can learn from numbers directly, whereas a few if no people can make sense out of giant matrices of numbers without any supplementary aids. By presenting the data graphically patterns can become more apparent, and the combination of the capabilities of human perception and the computational power of modern computers is a key to understand phenomenas around us. McCormick et al. also claims that 50% of the brain's neurons are associated with vision, so the appeal of having visualization systems is apparent. Most scientists and researchers use visualization as an invaluable tool to make sense of data. Proper visualizations can help discover trends, identify relationships and patterns, both obvious as well as non-obvious ones.

In [3], the visualization problem is characterized using three practical questions that are both sufficiently specific for researchers and easy to understand for practitioners: *What is presented?*, *Why is it presented?* and *How is it presented?* The *what* addresses the structure of the data in question, and *why* states the motivation of doing it, while the *how* answers what particular visualization methods will be used. All of these questions can aid the development of visualizations, and is important to have in mind when planning and designing them. Analyzing the problem domain and figuring out why you do something is as important here as in other research areas. As the ultimate goal of visualization is not to create pretty imagery or fancy animations, but to serve as a tool for humans to gain insight into data [40]. Schneiderman[39] introduces the Visual Seeking Mantra: "Overview first, zoom and filter, then details-on-demand", which serves as a guideline for how to design visualization that works in a variety of scenarios.

2.3 Visualization of Movement Data

Following the rise of commercial tracking equipment and the number of ways to do positional tracking, the area of visualization such data as been a research area of interest among many researchers. Research has been conducted at many at different scales ranging from flight data to animal movements.

Klein et al. [27] presents novel techniques for visualizing plane trajectories based on large data sets from Air Traffic Control in a scalable way both considering visual space and computational complexity. Although the scale is much larger than that of crowd movements at a university campus, similar techniques may be applicable to both.

An investigation of aggregation methods suitable for movement data were undertaken by Andrienko & Andrienko [5], wherein they also looked at what visualization and interaction techniques that can be used to explore massive movement data. They have considered different aggregation methods and contributed a general framework. The paper neglects implementation details, but suggest that most of the aggregations can be done using standard database functions. In another paper, the authors suggest a method for spatial generalization of and aggregation of movement data by transforming trajectories in to flows between areas and a method to spatially group the areas [9].

Chen et al. [14] presents an overview of visualization techniques in the context of traffic data through a survey of current research and a description of the visualization process pictured in Figure 2.1. Applications of traffic data visualization are classified into tasks: *Visual monitoring of traffic situations*, *pattern discovery and clustering*, *situation-aware exploration and prediction* and *route planning and recommendation*. For future research they suggest extending the work to social transportation.

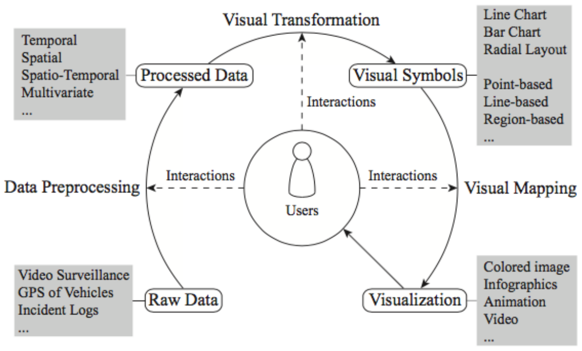


Figure 2.1: Pipeline of traffic data visualization [14]

The data provided by positioning systems are spatio-temporal in nature. That is, they describe objects existing in a geographical space changing over time. This fact makes analysis more complex, but makes the data useful for a variety of purposes, including but not limited to studying the different properties of a place, the dynamics of an event such as a large festival, social gathering, or the behavior of moving objects [6].

Von Landesberger et al. [45] focus on what different *spatial situations* look like at particular moments in time. They suggest different questions such as "how big are the flows between certain areas at a certain time?", or "which areas has many people at a certain time moment". Furthermore, they state that the temporal aspect can be analyzed to illustrate how spatial situations change over time.

2.4 Indoor Positioning Systems

In recent years, the research area of tracking people and objects have become increasingly more popular and interesting[36]. Outdoors the Global Positioning System (GPS) is available and provide an efficient and fairly accurate way to pinpoint objects. However, GPS does not work very well inside buildings or in areas packed with buildings such as in an urban environment[12]. There exists solutions based on both WiFi and Bluetooth that can track objects inside buildings[29]. NTNU uses the Cisco Mobility Service Engine (MSE) to provide the location analytics. The engine uses existing network infrastructure (here: WLAN access points) to aid the analysis. Triangulation requires the device to be within reach of at least three different access points, a requirement that is usually satisfied to the large amount of access points on campus.

Indoor positioning systems have a lot of potential use cases ranging from emergency control to increased revenue for business owners[12]. Other areas of application include, but is not limited to museums, logistics and optimization, industrial robots, guiding vulnerable people and environmental monitoring[32].

2.4.1 Usages of Location Data

Yin formulates the process of urban planning in three steps: *survey*, *analysis*, *planning* [47], where the first step is a data collection step, followed by analysis of the data and a final planning step where the results of the analysis is used for policy making to hopefully make better cities. A similar process can also be useful also for campus planning, as it can be seen as smaller scale city planning. The application described in this thesis is a tool to aid in the analysis phase by providing different views into data collected by other systems.

2.5 Potential Problems with Movement Data

According to Andrienko et. al [7], problems existing in any kind of data can be divided into three categories: *missing data*, *accuracy errors* and *precision errors*. They further specialize this categorization for movement data by incorporating *identities of movers*, *spatial positions* and *time references* as important data components. When any of these components are missing from a record, it can not be used for trajectory reconstruction. Each of the components may be subject to accuracy problems, while the position and time can have poor precision and the identifiers may be wrong. Precision refers to random errors caused by for example converting values into computer representations. These problems must be assessed when analyzing movement data and reconstructing trajectories from it.

Kandel et al. review challenges and opportunities associated with addressing data quality issues. They argue that data might be wrangled more efficiently by using interactive systems providing visualizations [26]. Datasets are prone to data quality issues such as missing data, inconsistent values, or unresolved duplicates, which can be a tedious process to assess. When identified, it is also not always obvious how to deal with the erroneous values.

2.6 Clustering Trajectories

Trajectories are a set of tuples containing spatial and temporal attributes, that can be interpreted as the path someone has taken. When showing many of them together they can often look intervened as objects often do not follow the exact same path. Research has been done on clustering such trajectories to get an aggregated view and to extract useful information from large sets of trajectories.

Andrienko et al. describes a procedure for extracting relevant places in trajectories to study place-related patterns and movements. The results can be for example be used to identify where people make long stops or where they meet and studying migration of animals [8].

Data-mining methods can be used to find frequent trajectories as studied in [38]. They propose the CBM method which divides an area into smaller clusters (squares), calculates the cluster count as the ratio between the number of coordinates of all trajectories passing through it and the cardinality of the dataset, and rejects inactive clusters below a certain user-specified threshold. The active clusters are further processed and linked to find frequent trajectories. Ferreira et al. [19] introduces a novel clustering technique named vector-field k-means using vector fields to induce a similarity between trajectories.

2.7 Modern Web Applications

The web as a platform has evolved rapidly in recent years and is being used for all kinds of applications apart from traditional web sites. The browser vendors and standards committees are working hard pushing the web forward by defining and implementing new APIs in a rapid fashion. The vast amount of available APIs and the ease of deployment have resulted in more applications running in the browser. At the same time, engineers have been working hard on improving the performance of JavaScript runtimes in the browser[1], playing a vital role in the evolution.

In recent years, as a result of the browser improvements, there has been a shift in how applications are developed. A lot of the work that has traditionally been performed on web servers and sent to the clients on request, have been assigned to the clients themselves. This evolution enables rich and powerful interactive applications running directly in the user's browser. Technologies such as WebGL has allowed the browser's to utilize more of the available hardware, including the Graphics Processing Unit (GPU) to create even faster and smoother experiences [35]. Software that were previously exclusive to the native platforms are now running directly in the browsers such as advanced 3D games and powerful word processing programs.

This paradigm shift makes installing software on own computers a thing of the past. Most software become ready for consumption directly in the browser, making things easier not only for the end user, but also for the developer who has better control of which versions of the software user's are running and can ship bug fixes and software updates more frequently without the user noticing.

2.8 Map Rendering

2.8.1 Existing Map Rendering Solutions

There exists different solution for maps on the web, both open-source and proprietary, libraries and ready-to-use solutions like Google Maps, CartoDB and ArcGis. Leaflet.js is the leading open-source interactive map library for web browsers. Mapbox is software company providing map services to users. Originally they provided a plugin to Leaflet.js called Mapbox.js, but now also provide Mapbox GL JS[22]. The latter is an improved library that uses WebGL for GPU assisted rendering and Web Workers for running processor intensive code off the main UI thread. Rendering times and perceived performance has been proved to be better when using the GPU [25]. Mapbox GL JS supports modern vector tiles out of the box, as opposed to Leaflet.js. MazeMap[33] is built on top of Leaflet and provide much better maps over the NTNU campus including rooms and other

[1]https://phoronix.com/scan.php?page=news_item&px=V8-JavaScript-5.4

facilities, which could have been nice to have. But Leaflet does not have adequate support for vector tiles, which was deemed necessary for this project. Therefore Mapbox GL JS was chosen as the map library. The static image tiles of the NTNU campus used by MazeMap can technically be used by the Mapbox renderer if they are made available from all hosts.

2.8.2 Tile-based Rendering

Tiles used in for map rendering are typically square shaped with a fixed size containing either compressed vector data or bitmaps, and are well-suited for long-term caching. Mapbox has defined a specification[2] for vector tiles, which among others has been adopted by Esri, a dominant actor in the Geographic Information System (GIS) industry[3]. The tiling approach has been widely used in web mapping software for a long time to reduce bandwidth usage when transferring data over a network, as only the tiles for the current viewport and the current zoom level need to be obtained. The concept of tiling is illustrated in Figure 2.2 and Figure 2.3.

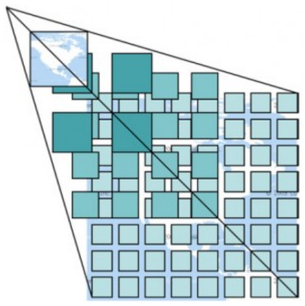
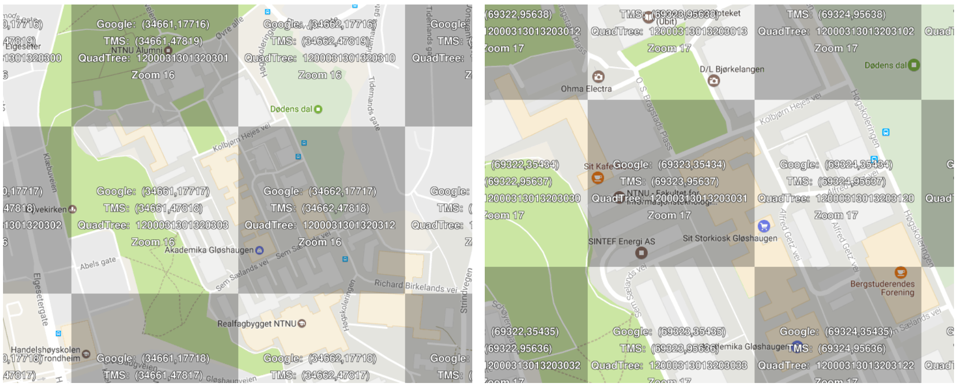


Figure 2.2: Illustration of a tiled map as a pyramid [15]

Contrary to static bitmap tiles that have traditionally been used in applications such as Google Maps, the vector tiles contain vector data instead of the raw pixels. This reduces the size of each tile, while also providing much greater flexibility in terms of styling. The client can be responsible for how exactly the end result should look, providing the ability to make adjustments in real-time. With modern computing devices having large amounts of processing power at their disposal, this is arguably a great benefit.

[2]https://github.com/mapbox/vector-tile-spec
[3]https://www.mapbox.com/blog/vector-tile-adoption/



(a) Z=16 (b) Z=17

Figure 2.3: Tile grid for different zoom levels [37]

In addition to the the vector tiles specification, there is file format called MBTiles. The file format specification describe an efficient format for storing millions of tiles in one SQLite database file. The output from Tippecanoe is a MBTiles file, which is a SQLite database in disguise. Therefore, this format can be read and understood by virtually any environment supporting SQLite, which is the case for most popular programming environments. MBTiles files are often referred to as **tile sets** [31].

2.8.3 Approaches

When rendering map data inside a web browser, there are essentially three different approaches that are used in the real world today:

1. Send all the features (typically as GeoJSON) to the client by request, and let the client take care of rendering them using the a rendering engine such as Leaflet or Mapbox GL JS.
2. Pre-render static images on the server at many different zoom level and send them as Portable Network Graphics (PNG) files to the browser on demand. The client can display these static images on top of the map directly.
3. Pre-render vector tiles on the server before sending the to the client. The client takes care of the actual styling of the features contained in the tile.

The first approach works really well when the amount of data is reasonably small. However, when the amount of data reaches a certain threshold, it will be time consuming to transfer all the data over to the client. Thus the client has to wait for all the data before it can even start painting something on the map. Recent versions of Mapbox indeed supports creating vector tiles on the fly in the browser from the JSON data, but you'll still have to wait for everything before anything shows up on the map.

Regarding how expensive it is to render the map, the second approach requires the least amount of work from the client. However, by using this inherently static approach, a lot of flexibility is lost in terms of how things will look on the map. The style must be predetermined and it is often hard to anticipate what would look best for different scenarios. For older computers this approach is sound, but with modern devices having lots of power there are better alternatives exploiting this power.

The third approach is a best of both worlds, hybrid approach. The client is responsible for the rendering, but the points are made available only when necessary. In Section 2.7 it was claimed that more and more processing work was moved to the browser, but some really resource intensive tasks such as crunching large amounts of raw numbers is still best executed in low-level programming languages on fast servers, as JavaScript is not optimized for this use case.

For this project, the following advantages were decisive for selecting the initial approach:

- It uses tiling, so the client only have to request the tiles currently in viewport and at the current zoom level. In addition, the tiles can be rendered in parallel and rendered as they arrive. Figure 2.2 illustrates how this works.
- The ability to adjust visual parameters are preserved, as the tiles are not static image files. Thus one can manipulate colors and sizes interactively in the browser without having to rebuild the tiles. This is important as different data sets require different visuals, that must be found through experimentation.
- The data fetched from the database will have to be cached anyway, so one might as well cache it as pre-generated tiles.

2.9 Tools, Protocols and Formats

2.9.1 GeoJSON

GeoJSON is a format for encoding a variety of geographic data structures[4]. The format can encode a wide array of common geometry types. It is has a standardized specification RFC7946[13] and is supported by many solutions from a number of vendors. Most of the mapping software suites that exist today has knowledge about the format and can render features defined in GeoJSON as shapes or objects on a map. Some of the most useful geometry types in the spec include `Points` (and `MultiPoint`), `LineStrings` and `Polygons`

```json
{
  "type": "FeatureCollection",
  "features": [{
    "type": "Feature",
    "geometry": {
      "type": "LineString",
      "coordinates": [
        [10.4018111136774,63.4185854963234],
        [10.4018111423982,63.4185834630124],
        [10.4018111543837,63.4194081529792],
        [10.4018111598974,63.4182065561121],
        [10.4018111742628,63.4179988291722]
      ]
    },
    "properties": {
      "name": "Some Trajectory"
    }
  }]
}
```

Listing 2.1: Example of a GeoJSON FeatureCollection

2.9.2 The WebSocket Protocol

WebSockets is a mechanism for two-way communication on the web[20]. In the traditional web using the normal HTTP protocol, a web server has accepted requests from clients (typically web browsers) and responded with some payload, meaning that only clients can initiate communication. WebSockets provide a bidirectional communication channel between clients

[4]http://geojson.org/

and servers by keeping a persistent connection between the two parties where both can send messages. The WebSocket Protocol is an independent TCP-based protocol.

A WebSocket connection is established by a client sending a regular HTTP request with an `Upgrade` header indicating that it wants to establish a WebSocket connection. This process is known as the WebSocket handshake. Servers supporting WebSockets will respond with an agreement through at HTTP response with status code `101` and an upgrade header. After the handshake, the two parties can send messages back and forth without the overhead of HTTP[46].

This mechanism is useful in many applications, for example where the server needs to inform the client about progress on the server without the client explicitly asking. It is also useful in chat or messaging applications were clients continuously must be notified of new messages and people entering and leaving.

2.9.3 Tippecanoe

Tippecanoe[30] is a program written in C++ that, given a set of points, lines or polygons, attempts using clever algorithms to simplify these primitives by skipping points while maintaining a similar visual output. It does so by looking at which pixels the primitives would cover and remove overlapping based on a z-ordering[5] of the points [21]. The program output is an MBTiles file containing data of different vector tiles at various zoom levels specified as arguments to the program. The space savings of using this program can in some cases be substantial.

2.9.4 PostgreSQL and PostGIS

PostgreSQL markets itself as *The world's most advanced open source database*[6], and is as the name and slogan implies a Structured Query Language (SQL) database with a large set of features. PostGIS is an extension to PostgreSQL providing support for geographic objects, allowing location queries to be executed in an efficient manner. PostgreSQL is suitable for many different use cases and applications, while also being easy

[5]https://en.wikipedia.org/wiki/Z-order_curve
[6]https://www.postgresql.org/

to interact with from a developer perspective through the powerful query language SQL. Combined with PostGIS it provides a stable foundation for building applications that handles spatio-temporal data. PostgreSQL is a traditional RDBMS with row-based storage with good vertical scalability.

2.10 Web Application Performance

In a web application, there are basically two important factors that affects the performance the application: Transferring code and data over the network from the server to the client, and running code and rendering the page on the client (run-time cost). The latter is most crucial on mobile clients, but also worth taking into consideration on desktops. The amount of code and data that needs to be transferred can be reduced by using various compression methods, while execution times can improved by writing more efficient code. For desktop computer connected to stable high-speed networks, the data transfer is not too important until it becomes very large. But on mobile networks where stability and speed varies, and data plans can be costly, it is very important. The intended usage scenario for this project is on desktop computers, so it is not the most crucial aspect for the application code.

There are many different categories of applications. Some applications are large by themselves having lots of JavaScript code that must be run on startup. Other applications are smaller in sheer app size, but must load and handle massive amounts of external data. An application visualizing pre-processed data sets fall into the latter category. Granted, when maps are involved, a map renderer is required. Map renderers are often extremely complex and needs to do lots of projections and calculations. Anyway, how fast the latter type of application is depends largely on the size of the raw data loaded into it. Occasionally, the cardinality may be excessive and compression can only help with the actual data transfer from A to B. When it has arrived at the destination, the data must be decompressed (lots of CPU cycles spent) and visualized. There is a limit to how much uncompressed data the client can handle, but modern hardware can perform well on even large datasets (especially when GPUs) are employed. Rendering graphics is orders of magnitude faster on GPUs than on the Central Processing Unit (CPU), due it is parallel nature.

Chapter 3

Research Methodology

This chapter will describe how the research leading to this thesis was conducted, starting with a description of the research method, followed by a presentation of the research questions guiding the research and how the results will be evaluated.

3.1 Research Method

The research methodology used in the project resembles the *Design science research* approach. Design science research is a set of techniques for conducting research in Information Systems (IS). The ultimate goal of design science research is to improve and understand certain aspects of information systems. Achieving this goal involves gaining knowledge by creating artifacts and analyze the use and performance of these. Artifacts can come in many forms including algorithms, user interfaces and system design methodologies[44].

Takeda et al proposed a 5-step process model illustrated in Figure 3.1. Table 3.1 gives a description of each individual step.

The problem awareness for this project is described in Background and Motivation section of this chapter. The suggestion phase aims to be a creative step where possible solutions are envisioned based on previous work and other research as described in Chapter 2. This step is closely related to the *development* step, where the ideas from the preceding step is being

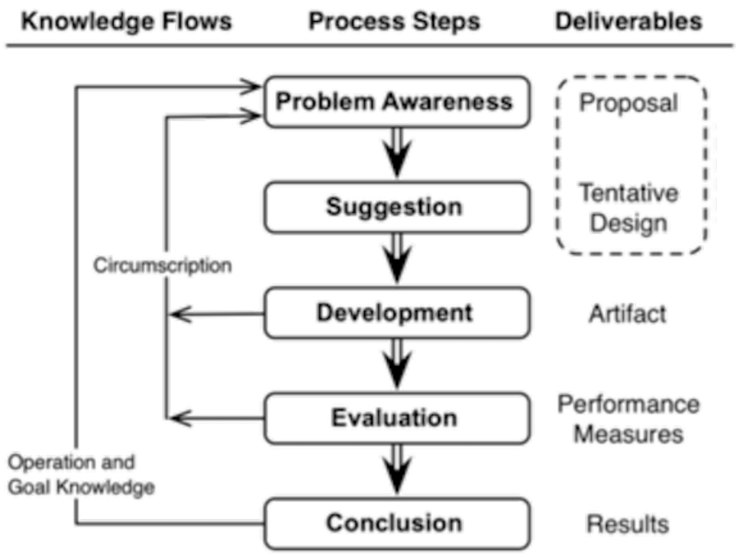


Figure 3.1: 5-step cycle [44]

put to life. The development step is discussed in Chapter 5. Following the development is an evaluation step. Here the artifact should be evaluated according to some given criteria. The criteria for this project is a set of requirements defined in Section 4.3. The conclusion is the last step. If the results were found satisfactory, this is typically the final step of the cycle. Otherwise the cycle might be repeated. The conclusion should clearly communicate the knowledge contribution of the research and any loose ends that could possibly be topics for future work. Contribution of knowledge is a vital part of the design science research approach for work to be accepted as research and not only development.

3.2 Research Questions

The research investigates the adequacy of the web platform in building visualization applications for large datasets. Further, it utilizes the developed artifact to possibly discover new patterns or confirm known ones, and hopefully detect patterns that could occur due to flaws and anomalies in the recorded data.

- **RQ1**: What are the limits of the web platform and PostGIS for

Step	**Description**
Problem Awareness	Finding an interesting problem in business, society or science
Suggestion	Suggesting a possible design solution in the form of an artifact
Development	Implementation of the artifact
Evaluation	The artifact is evaluated according to predefined criteria
Conclusion	The end of the research cycle

Table 3.1: 5-step process model proposed by Takeda et al.[41]

building data-heavy visualization applications?

- **RQ2**: Is it possible to build a reasonably performant visualization using existing web maps and database technologies?
- **RQ3**: What kind of patterns can be revealed and confirmed by a human research utilizing the application?
- **RQ4**: Can visualization methods different from the ones used in previous work give new insights?

3.3 Evaluation Criteria and Results

The artifact will be evaluated according to measured characteristics of its performance and its ability to provide useful visualizations to the user. This will be measured by performing benchmarks for different usage scenarios. The main knowledge contribution of the research will be a presentation of some interesting patterns revealed by the systems that could for example be of use in future work on assessing data quality and the results from the performance evaluation.

Chapter 4

Problem Elaboration and Requirements

This chapter extends the introductory motivations by giving deeper insight into the problem area to which this application is tailored and what kind of usage scenarios one can imagine. To aid in this elaboration, various personas and usage scenarios are described. In the final section, the requirements of the system are presented.

4.1 Persona

Personas for each usage scenarios are often created to gain a deeper understanding of the *users* of a system and are used to improve usability. The persona is a fictive character with attributes and skills constructed with the purpose of imitating a real-world scenario [34].

4.1.1 Facility Manager Mikkel

Mikkel is 40 years old and works a facility manager responsible for buildings and allocation of space at the campus NTNU. Part of his job is to plan, build and evaluate areas on campus dedicated to student work such as “Drivhuset”, regular reading rooms and other co-working spaces to make sure the students’ needs are met and that the campus is a nice place to be.

4.1.2 Technical Janitor Pål

Pål is 30 years old and works as technical janitor. A considerable part of his daily work is to make sure there is decent Wi-Fi connectivity everywhere. He and his colleagues maintains thousand of access points and replaces and installs them frequently.

4.1.3 Analyst Susanne

Susanne works as a researcher with special interest in movement patterns and how the spatial positions of different actors varies with time. Luckily for her, modern tracking equipment installed everywhere provide her with large datasets. She is interested in finding flows of people at a specific place, in addition to insight into the origin and destinations of people's trips. Susanne is a very experienced analyst and has worked with many different visualization tools throughout her career.

4.2 Scenarios and Use Cases

A clear vision of how and for what an application is going to be used is often useful in the process of specifying the requirements and designing an application. A story involving a person and a description of the task at hand, and how the application can aid in solving the task are often invaluable in this process.

4.2.1 Movement Between Areas of Interest

Susanne is doing research on mobility patterns and would like to find out how people are moving through the campus during course of a day. In this particular case, she wants to see how many people is moving from the campus to the training center. She opens the application and chooses *Buildings* in the sidebar to enable buildings mode. Next, she chooses the desired time slots 08:00-10:00, 10:00-12:00, 12:00-14:00 in the form and click the generate button to build the visualizations. When each visualization is finished, she clicks the notification bubble to open the visualization in a new tab. In each tab, she will see curved lines connecting buildings, and circles on the buildings on which she can hover the mouse a see the number of people moving out of it. To compare the different time slots she switches back and forth between the tabs.

Id	1
Name	Movement Between Areas of Interest
Description	Visualize movement between defined areas, particularly buildings.
Actors	Researchers
Assumptions	The application is started. Time range and area of interest decided.
Steps	1. Select Buildings from the tab bar 2. Select the desired time range 3.Click generate 4.Wait for the visualization 5.Interact with the visualization and look for curves going to the area of interest.

Table 4.1: Movement Between Areas of Interest

4.2.2 Detect Tracking Errors

Pål is suspicious of the accuracy of the tracking system installed and wants to see if he can detect any weird patterns in the tracking data. Therefore he opens the application and generates a collection of snapshots for arbitrarily chosen time ranges. He opens them in different tabs and starts zooming in and moving around on the map to see if something looks unexpected, and if these patterns are present on many of the snapshot. After some exploration he finds a pattern that cannot possibly be right: People seems to me flying in the air. Following his new findings he initiates an investigation of the tracking equipment in that area.

4.2.3 Check If Areas Are In Use

The facility manager Mikkel opened a new co-working space for students a couple of months ago. The new student area was a prestige project for the faculty owning it. Now he would like to evaluate if it had become a success and people were using it by looking at usage patterns throughout a day.

Mikkel opens his web browser on the computer in his office and starts the application. He would like to know how the usage patterns changes between

Id	2
Name	Detect Tracking Errors
Description	Try and figure out possible tracking errors
Actors	Maintenance People
Assumptions	Tracking errors suspected, application opened
Steps	1. Select a time range in the suspected period 2. Generate snapshots for these. 3. Repeat 1. and 2. for different ranges. 4.Compare and look for suspicious patterns.
Post Condition	Go and fix the faulty trackers.

Table 4.2: Use Case 2: Detect Tracking Errors

time intervals throughout the day: 8:00-10:00, 10:00-12:00 and 12:00-14:00. To accomplish his task Mikkel creates snapshots for each interval using the tile set generator and waits some seconds for them to finish. Then he opens up each interval in three different tabs and switches between them. He notices something he doesn't quite expect: It is barely used at all between 10:00-12:00, but it does have some usage both before and after. Intrigued by the discovery, he repeats the process for a couple of other days as well. However, from what he sees, it does not seem to be a recurring pattern.

4.2.4 Maintain Decent Wi-Fi Connectivity

Pål's job is to make sure the Wi-Fi connectivity is acceptable throughout the campus. This entails identifying bad access points and installing new ones. One day he gets a telephone from someone claiming the Wi-Fi reception is bad in a particular area. Knowing that there exists a tracking system that uses the Wi-Fi access points to track devices, he can use point cloud visualizations in the application to identify areas with weak connection.

He opens the application and generates a snapshot for the current time, selects it and moves the map to the reported area. There he sees some weird patterns and empty areas in what is known to be popular places, indicating that the tracking is bad and goes out to investigate it. It turned

Id	3
Name	Check If Areas Are In Use
Description	Assess if new facilities are being used as desired by the creators.
Actors	Building and facility managers
Assumptions	Desire to figure out if areas are used
Steps	1. Select one-hour time ranges throughout the day. 2. Zoom in to the desired area. 3. Compare the different time ranges and look for emptiness.4. Repeat for new time slots.

Table 4.3: Use Case 3: Check If Areas Are In Use

out the caller was right, some of the access points in that area were indeed broken. Thus he must replace them, so the reception can be restored to acceptable levels.

4.3 Requirements

In order to make a system suitable for fulfilling the desired needs, some requirements must be established up front that will guide the development. Most of the requirements are based on suggested future work from prior projects. The requirements were fluid and iterated upon throughout the process, but the base requirements were fixed. The goal of the project is to create a useful tool for doing research on positioning data and the defined requirements are formulated with this in mind.

4.3.1 Functional Requirements

FR1 The application should provide a web-based interface with an interactive map to explore positioning data.

FR2 It should be possible to browse all the available data collected during the two-month period 02.09.2014 to 02.11.2014.

FR3 It should be possible to select data from multiple continuous time ranges such as 15 minutes, 1 hour, 2 hours, 8 hours, or a week,

Id	4
Name	Maintain Decent Wi-Fi Connectivity
Description	Use missing tracking data to confirm broken access points.
Actors	Maintenance People, Students
Assumptions	A student has reported bad Wi-Fi connectivity
Steps	1. Select time ranges before and in the reported period. 2. Zoom in to the reported area. 3. Look for missing data 4. Make a conclusion based on the visualization
Post Action	Replace the broken equipment

Table 4.4: Use Case 4: Maintain Decent Wi-Fi Connectivity

and aggregated time periods such as "each Monday at 12" by using appropriate controls in the interface.

FR4 The user interface should allow the user to adjust the visual parameters of the map, such as blur opacity and radius to accommodate different dataset sizes.

FR5 The application should provide insight into the data from different angles, by having multiple modes of visualization for showing both static positions, overall movements and movements between objects of interest.

FR6 It should be possible to show the positioning data a point cloud on the map to give an overview over all tracked devices. Additionally, it should be possible to distinguish the points visually based on the floor on which the device was tracked.

FR7 There must be a simple way to compare data from different time periods by switching back and forth between datasets.

FR8 One should be able to see some aggregated numbers regarding the viewed data.

FR9: The user should be informed by the system when a visualization is finished processing and ready for use.

4.3.2 Non-functional Requirements

NFR1 The delay between requesting the generation of a tile set in the browser and the corresponding visualizations becoming explorable in the application should be less than a few seconds for hour-long ranges.

NFR2 It should be ready to present even more data as it becomes available at a later time, and not be restricted to the currently available data.

NFR3 Where possible, open standards should be used to ease integration with other systems.

The main goal of the application itself is to provide the necessary tools for researchers and other stakeholders who have interest in exploring and getting more insight into the captured data.

Chapter 5

The Application

In this chapter the final solution is presented. First the entire system, its architecture and how it works is thoroughly described along with design decisions taken. Further, the user interface of the system is presented with descriptions of how an end user interacts with the application. In short terms, the application is a web based exploration tool that allows an individual to explore a positioning dataset with a focus on comparability.

5.1 Database

Originally, the positioning data was available as JSON files and in an AsterixDB instance from previous work. However, it was early on in the project found necessary to have the data reside in a database more suited for this application. Therefore, all the available data ranging from September to November 2014 was copied from AsterixDB into PostgreSQL and indexed according to the presumed data access patterns of the application. In addition, a new column named `location` was added to store the geometry representation of the `(latitude, longitude)` pair of the recording to take advantage of PostGIS capabilities. After all the data was migrated over to the new database, it was ready to be used by the application. Figure 5.1 shows how the recording is distributed over hours of the days and provides an indication of days for which queries can yield large result sets. All the tracking data reside in one single table. Thus, the data is not normalized according the traditional RDBMS principles [17], but for this application speed is crucial and `JOIN`s are avoided as they are theoretically slower than not joining.

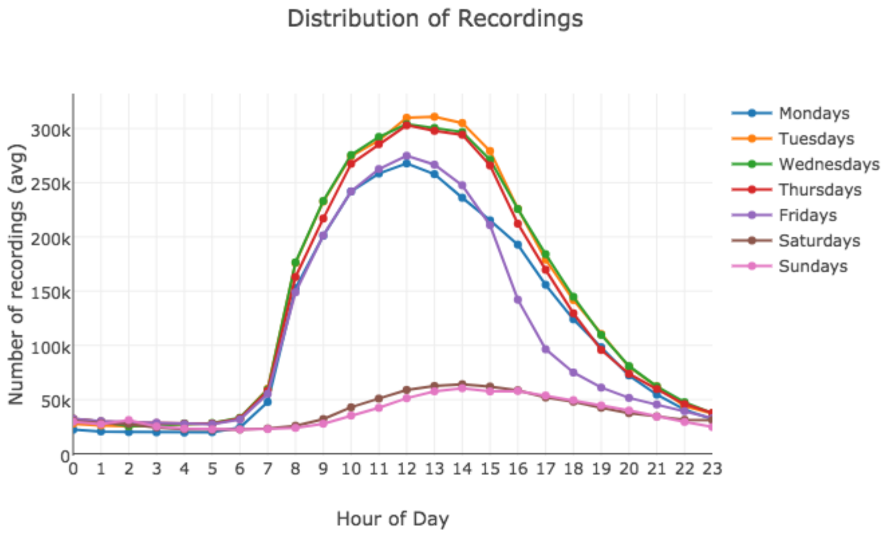


Figure 5.1: Distribution of data per week day per hour

The indexes created are listed in Table 5.1. Running the Postgres command `EXPLAIN ANALYZE` to see how the query was planned and executed by the database engine revealed that these were actually used to speed up retrieval. However, indexes does not come without a cost and leads to increased space requirements due to their possibly large sizes. Table 5.1 also enumerates the size of the current database including the size of the indexes. One can see that they require quite a bit of space, but the trade-off was found to be worthwhile due to increased retrieval speed.

5.2 Technical Design and Architecture

The system is designed using a Client-Server paradigm where communication is facilitated via WebSockets. This protocol enables two-way communication where both the client and the server can send messages, as opposed to the classical request-response cycle in the HTTP/1.1 protocol. Being able to send messages both ways is a great ability when the server has to take care of long-running jobs and must be able to notify the user when something has happened, such as completing a longer task initiated

Name	**Type**	**Size**
idx_day_hour	index	4195 MB
idx_device	index	4517 MB
idx_device_day	index	6897 MB
idx_floor	index	12 GB
idx_timestamp	index	5714 MB
locations	table	47GB
locations_pkey	*index*	*9848 MB*

Table 5.1: Size of tables and indexes

by an earlier client request. An explanation of how WebSockets protocol works is given in Section 2.9.2.

The developed system is a full-stack implementation of a complete system dealing with everything from an interactive web user interface, managing network transfer, to orchestrating the fetching and processing the data recorded by the tracking systems on campus. Therefore the system follows a layered approach and is divided into multiple modules that communicate using standard mechanisms. The first being a user facing web application running in modern desktop web browsers and the second is a server back-end responsible for handling requests from the user and keeping the client up to date with new information from the server. Together with the database, and the worker processes running on demand they form a complete system for interacting with the position data. Figure 5.3 illustrates the general architecture of the system and its components.

5.2.1 Web Application

The web application (also referred to as the client) is the user facing part of the system. It contains all the user interface components used for interacting with the system and a large map that is used to display the visualizations to the user. The client is written in JavaScript using Facebook's open-source React[1] library for declarative user interfaces along with Cascading Style Sheets (CSS) for styling. It uses Mapbox GL JS for map rendering as discussed in Section 2.8. The application code is written in multiple files using ECMAScript 6 Modules and stitched together using the module bundler Webpack (See Appendix C). When the application is

[1]https://facebook.github.io/react/

launched in the browser, a long lived WebSocket connection is established with the server - opening up a channel for bi-directional communication. A thorough walkthrough of the user interface is given in Section 5.3.

5.2.2 Back-end Server

The back-end of the system, which runs on a server separated from the client app, consists of multiple heterogenous processes working separately on a single responsibility. The entry point to the backend is the Node.JS API server that is placed in front and both accepts HTTP requests and manages WebSocket communication with the connected clients. It does not do much raw processing on its own, but is responsible for routing and initiating the correct child processes. In layman's terms it acts as a front desk manager for the rest of the system. The system is currently set up with only one of these, but with the use of a Load Balancer in front, multiple instances of the API server, can if deemed necessary, be launched to handle more requests concurrently. Node.js as a technology was chosen because of the researchers experience with it, the fact that it has a lot of third-party libraries available via Node Package Manager (NPM) providing great support for technologies such as WebSockets, Redis and PostgreSQL. In addition, it has a fast development cycle making it easier to iterate on the artifact and adapt to feedback.

Tile Generation Requests A typical task for said API front controller is to receive *Tile Generation Requests* from a connected client, parse the request messages, create a job object with the extracted parameters, and put the object in the job queue using `queue.create()`. After the job is created, it responds to the client with a message saying that the request has been enqueued. In the event of a job being reported as either finished or failed by the queue system, the web process will forward these messages to the client over the WebSocket connection.

Serving Map Tiles Another obviously important task is to serve the correct map tiles when requested by the map renderer. This is performed by looking up the correct tiles based on the tile set name and the given coordinates (see Section 5.2.3) from the tile sets stored on the local file system, and send the *protocol buffers*[2] contained in the MBTiles back to the client.

[2]https://developers.google.com/protocol-buffers/

The API process merely puts job requests in the queue, but does not actually execute them. Running jobs in the queue is done by a set of *worker processes* that work independently of each other. These workers pops items from the central job queue managed by Kue when they are available (i.e. not processing another job) and the queue is not empty, figure out what type of job it is by looking at the `type` attribute and processes it accordingly. When a job is finished, the queue notifies the master process and the worker becomes ready for new tasks. This design ensures that multiple long running tasks can be run in parallel and even restarted on failure. The number of concurrent tasks is limited by the number of available workers. Running multiple workers is managed using the standard `cluster` module of Node.js, and the number of processes is set to the number of cores of the system it is running on.

The most important job in the system is arguably the `make-tile` job. This job type fetches data from the PostgreSQL database based on parameters given by the user, runs the data through the tile generation pipeline and outputs the tile sets to the file system, where they are persisted and available for future use without needing to re-run the same process for tiles built with the same parameters. The last statement is an important aspect with the current design. Even tough some visualization may take a bit of time to generate, they will be instantly available for future use. Figure 5.2 illustrates the tile generation process as a sequence diagram.

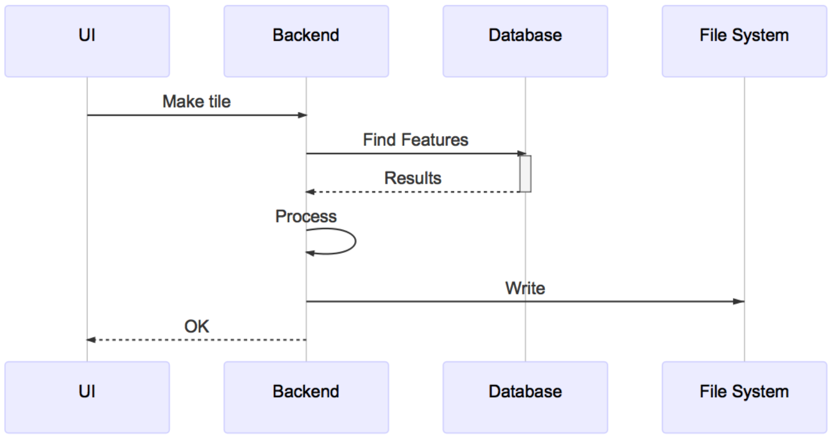


Figure 5.2: Tile Generation as a Sequence Diagram

A job is defined as an asynchronous JavaScript function that takes a `job-`

Object and a `done` function as arguments, as required by the queue system Kue[3]. The function is called by one of the worker processes. Upon finishing whatever the function is doing, `done` is called to indicate completion. The callback is called regardless of result with the first argument being an `Error`-object on failure or `null` on success. Kue will mark the job as either completed or failed in the Redis store accordingly.

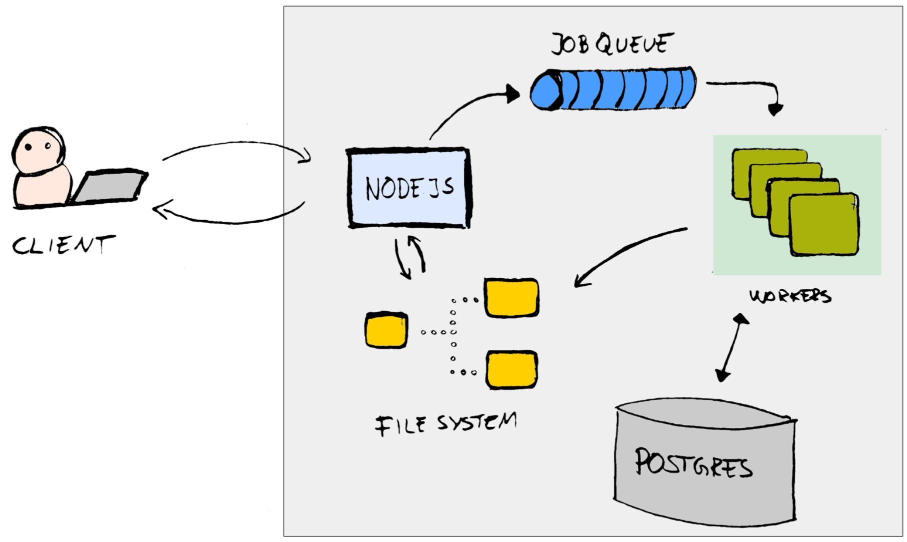


Figure 5.3: General Overview of the System

API Written in Node.js and manages other processes. It is responsible for client communication.

Workers Concurrent processes that executes functions based on tasks available in the job queue. These functions are also written in Node.js, but can spawn child processes when necessary.

Job Queue Notifies workers of available jobs. The queue stores the jobs and their statuses in a Redis database. This ensures that jobs are persisted and is not lost if some processes dies or must restart.

Because some of the tasks may take a rather long time to finish, it is imperative that the system is able to notify the user when work is either

[3]https://github.com/Automattic/kue

completed or has failed. A basic user test showed that the user would have no idea what was going on unless there was some kind of confirmation or notification. This goal is achieved by having the server send notifications to the client at appropriate times during the lifetime of the job, which the web application then can show as messages popping up in the top-right corner of the interface.

5.2.3 How the Tiling Works

The map rendering engine on the client expects that there are ready-made tile sets in the MBTiles vector format available in order to show information on the map. These tiles are generated per user's request and cached on the file system for future (re)use. Following is a description of how the process works.

Tiles are generated for many zoom levels, but in this application it is limited to the range 14 to 20. As the available dataset only covers a small geographic area, the lower zoom levels is neither useful nor particularly interesting. The tiling scheme used by common mapping software supports zoom levels 0 to 22, where 0 covers the entire world and contains one tile, while increasing zoom levels provide more detail and exponentially increasing numbers of tiles given by 4^z for zoom level `z`. Mapbox uses a tile size of 512×512 pixels. Other systems may use other sizes such as 256×256 pixels.

The map renderer needs to know where it can obtain a single tile from the tile sets. The URL scheme expected has parameters for `z`, `x` and `y` to look up a specific tile based on zoom level and coordinates. In addition the URL has a `name` parameter to know which tile set to use. The full URL format for tile lookups is `http://{d}.domain.com/{name}/{z}/{x}/{y}.pbf`. The server can be configured to treat all subdomains `d` as the same thing. This is done to bypass a limitation of the number of concurrent HTTP requests the browsers send to a single domain, so more tiles can be fetched in parallel. When the user is interacting with the map and has chosen the tile set named `name`, Mapbox GL will automatically send HTTP requests to this URL to find the tiles matching the current browser viewport.

Lower zoom levels presents the challenge of a cluttered view where many points are overlapping. To take advantage of this to reduce file sizes, Tippecanoe can optimize for different zoom levels by reducing overlapping data using different heuristics. By default, it will try to limit each tile to 500 kB or to a maximum of 200,000 features. The impact of utilizing such optimizations is discussed in Section 6.2.2, but in short it provides a trade-off between resource requirements and the level of accuracy at each individual zoom level.

5.3 Description of the User Interface

The main part of the user interface is the map view that covers most of the viewport, and a sidebar where the user can control certain aspects of what is shown in the main view. The user interface supports multiple tabs similar to what is now commonly implemented in web browsers, allowing multiple datasets to be examined at once, with instant switching between them. The zoom and map position is synchronized between the tabs for easier and more meaningful comparison.

5.3.1 Visualization Modes

The application features three different visualization modes, that have their own unique way of visualizing the data. The different modes can be chosen by selecting the desired mode in the tab bar located at the top of the sidebar. The visualization modes available are

Snapshot Snapshot is the default mode and shows recorded locations as a point cloud on the map.

Movement The Movement mode shows the recorded trajectories of all devices. Instead of painting dots at the exact locations, it paints lines between recorded device locations.

Buildings This mode shows movements between buildings (origin-destination matrix) as curved lines between buildings along with numbers of devices that has been recorded when hovering over specific buildings. The lines are colored based on the relative frequency of movements. A red line indicates a high frequency of movements, whereas green lines indicates a less frequent passage.

From a user's perspective, the different modes are similar in terms of interactions, but they have slightly different adjustments and a very distinct graphical appearance on the map. A more detailed description of how the different modes are processed is given in Section 5.4. The system can be extended with even more modes by implementing two modules: A function for retrieving required data on the server and a React component for displaying the data on the client, following the contract used by the existing modes. Figure 5.6 illustrates the visualization output of the modes.

5.3.2 Building New Tile Sets

Below the tile selection box there is a form containing input fields for creating new visualizations. Here one can choose time parameters, either as a **continuous time range** (14.10.2014 12.00 - 14.00) or as an **aggregate** over a period such as "Mondays at 12pm". In addition, one can choose to give the visualization a name to make it easier to find later in the list. By submitting the form, a new *tile generation request* is sent over the WebSocket to the server. If the tile set already exists based on a hash value of the parameters, the job will not be executed and the user notified of its existence.

The range picker is built using three components: One for choosing the date, one for the range size (`5m`, `30m 1h`, `2h`, `4h`, `8h` and so on) and finally one for picking the segments in that range (`12 to 20` when the size is `8h`). The reason for this design is that it should be quick to generate consecutive ranges by clicking the arrow. An example of this is choosing the three ranges 14.10.2014 11.45-12.00, 12.00-12.15 and 12.15-12.30 by two consecutive clicks on the next button.

5.3.3 Adjusting the Visual Parameters

All the different visualization modes support visual adjustments, which lets the user alter the appearance of the visualization by changing parameters used by the renderer. The available options depend on the current context. Due to the differing number of points in different datasets, having the ability to adjust parameters such as opacity and radius was deemed necessary. Otherwise, sets with less data would barely be visible, whereas larger datasets would become too packed. By adjusting the parameters themselves, the users are able to manually find the right balance for the dataset in question. A reset button restoring the values back to default is also provided. The adjustments are implemented by dynamically changing the

Table 5.2: Visual adjustments for each mode

Snapshot	Circle Radius, Blur, Opacity, Color
Movement	Line Width, Blur, Opacity, Color
Building	Line Width

paint properties of the layers according to the Mapbox Style Specification[4] in response to user interaction. These adjustments are made possible by using vector tiles as discussed in Section 2.8.3. The available adjustment options are enumerated in Table 5.2. It is also possible to change the background map to have a dark-on-light visualization instead of the default light-on-dark.

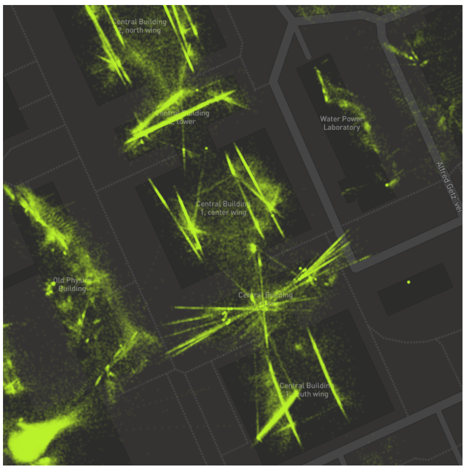
5.3.4 Distinguish Between Floors

When a tile set has been chosen by the user in the select box of the sidebar, it will immediately show up on the large map view. Here the user can interact with it by *zooming* in and out or *dragging* to alter the viewport (move around on campus). On the right there is a button labeled "Floors" that, when clicked, will open a popover panel with a floor selector and an option to distinguish between the floors in the visualization. When the distinguish option is enabled, data points will be grouped by the floor in which they were captured and rendered as different colors matching the background colors of their corresponding toggle switches. By toggling single floors, the user is able to show or hide specific floors depending on the current needs. To make floor management faster, there is also buttons labeled **All** and **None**, that will enable and disable all the floors respectively. Selecting floor is currently only implemented for the snapshot mode, since there has not been found a good way to visualize movement between floors. Figure 5.4 illustrates how the view changes by using the floor picker.

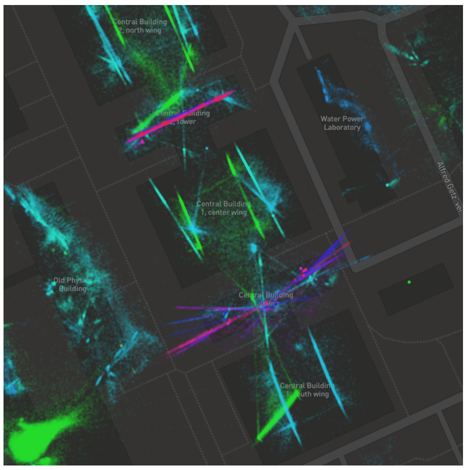
5.3.5 Comparing Tile Sets

To aid in comparison of tile sets, one can open and load multiple tile sets and once and switch between them nearly instantaneously. Figure 5.5 shows it in action with three open tabs labelled the same as the name of the respective tile sets. When zooming or moving the map in one tab, it will

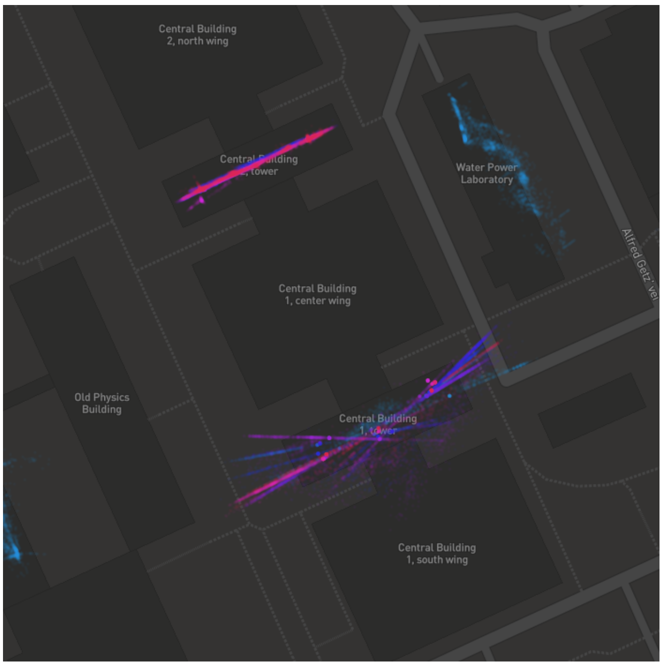
[4]https://www.mapbox.com/mapbox-gl-js/style-spec/


Central Building 2, north wing
Central Building 2, tower
Water Power Laboratory
Central Building 1, center wing
Old Physics Building
Central Building 1, tower
Central Building 1, south wing
Alfred Getz' vei

(a) All the same


Central Building 2, north wing
Central Building 2, tower
Water Power Laboratory
Central Building 1, center wing
Old Physics Building
Central Building 1, tower
Central Building 1, south wing
Alfred Getz' vei

(b) Different colors


Central Building 2, north wing
Central Building 2, tower
Water Power Laboratory
Central Building 1, center wing
Old Physics Building
Central Building 1, tower
Central Building 1, south wing
Alfred Getz' vei

(c) Some floors hidden

Figure 5.4: Distinguish floors by painting different colors

be applied globally so all tabs have the same map position. New tabs can be opened by pressing the “Open Tab” button. Each tab has its own map instance with the necessary layers loaded into memory enabling instant switching.

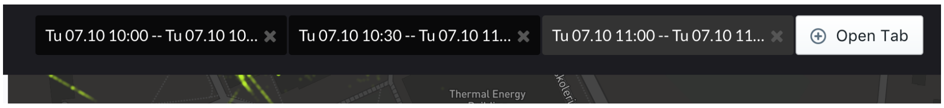

Tu 07.10 10:00 -- Tu 07.10 10...
Tu 07.10 10:30 -- Tu 07.10 11...
Tu 07.10 11:00 -- Tu 07.10 11...
Open Tab
Thermal Energy

Figure 5.5: The tab bar with three open tabs

5.4 Data Processing

This section documents how the data goes from the database, through a simple processing pipeline, and finally becomes available to the map client for each of the supported visualization modes.

5.4.1 Snapshots

Snapshots are more or less direct representations of the recorded locations in the dataset visualized as a point cloud. The recordings are fetched from the database as GeoJSON directly using the powerful JSON capabilities of PostgreSQL shown in Listing 5.1, grouped by the floor to which the recording were assigned by the tracking system and fed directly into the tile generator. Each floor will become a separate layer in the final vector tiles. The layer naming is important as they need to be known by the client code. Tippecanoe will layer the points based on the `layer` property of the `tippecanoe` object, and the same name is used in the client to determine which layers to show and hide when the floor toggling is used. The scheme used is the value of the `floor` column prefixed with `floor_`: `floor_1.etasje`.

5.4.2 Movement

The movement mode visualizes the trajectory of multiple devices over a defined period of time. There is not a one-to-one mapping between individuals and their devices, as they may have multiple devices, which results in some trajectories possibly being boosted. The trajectories are created by aggregating the recorded device positions for each device over the

```sql
SELECT row_to_json(d) as result
FROM (
  SELECT 'FeatureCollection' as type,
  (
    SELECT array_to_json(array_agg(row_to_json(f)))
    FROM (
      SELECT 'Feature'::text as type, (
        SELECT row_to_json(z)
        FROM (
          SELECT CONCAT('floor_', floor) as layer
        ) z
      ) as tippecanoe,
      '{}'::json as properties,
      ST_AsGeoJSON(ST_Multi(ST_Union(location)))::json as geometry
      FROM locations
      WHERE created_at >= $1 AND created_at < $2
      GROUP BY floor
    ) AS f
  ) as features
) AS d
```

Listing 5.1: Select snapshots as GeoJSON

given time period. Single devices may have been recorded multiple times at approximately the same position, indicating lack of movement. The final artifact uses the built-in `ST_RemoveRepeatedPoints` of PostGIS to remove spatially close points. This method may not always be satisfactory for larger time ranges, as people may have returned to the same point after some time. Figure 6.7 shows the effect of attempting to filter points based on time and distance differences to smoothen the trajectories and removing similar points, which could be used in conjunction with clustering to retrieve popular paths.

The visualization produces a rather cluttered view of the trajectories by painting all of them using `LineStrings`. However, in areas with many overlapping trajectories one can see the trends of where the movement is happening. Between buildings devices are recorded only sporadically, with possibly long distances between points, so it is hard to infer paths taken. In [12], movements were map-matched to view-finding requests from MazeMap to infer a possible path. This mode does not currently employ a clustering technique as discussed in Section 2.6, but it can be implemented as part of a future work. In any case, this visualization is able to provide a view of a large set of trajectories for a selected time range. The view is in essence similar to the snapshot view, but instead of painting the point cloud of recorded positions, it paints the device trajectories as lines.

5.4.3 Building-to-Building

The method for extracting building to building movements relies on some aggregation features of PostgreSQL as well as some grouping in the application code. First, all trajectories for the given time range are selected from the database as in Listing 5.2. The returned rows contain the device id and its trajectory of recorded points along with the building name associated with the recordings, ordered by the time they were sampled.

```sql
SELECT
  device_id,
  (
    SELECT array_to_json(
      array_agg(
        row(longitude, latitude, building, created_at)
        ORDER BY created_at
      )
    )
  ) AS trajectory
FROM locations
GROUP BY device_id, salt_timestamp
```

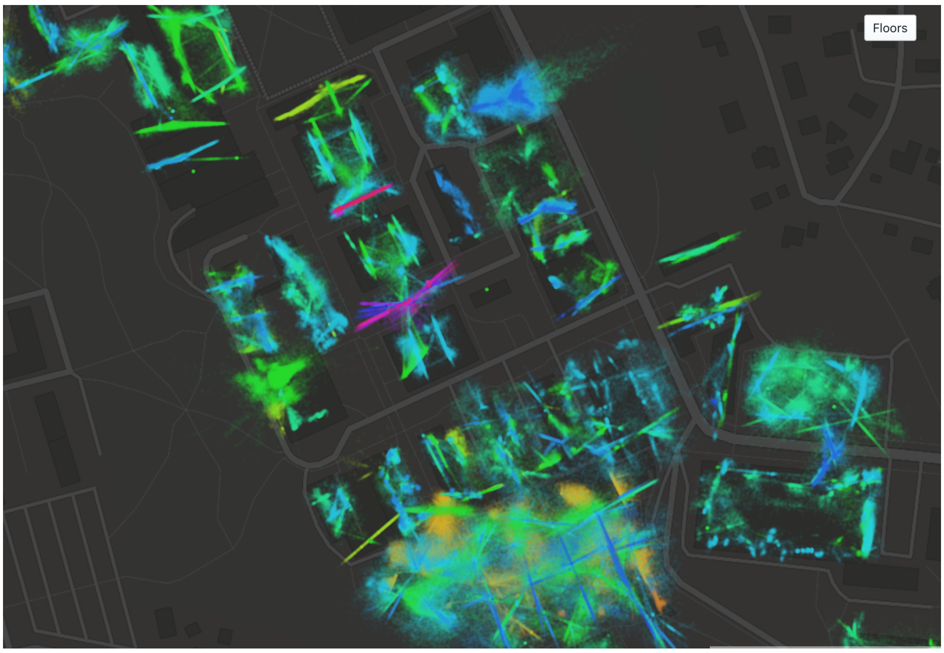
Listing 5.2: Select trajectory of each device

Next, the rows are iterated through and each trajectory is flattened using a simple heuristic: Only keep the records for which the next building is not the same as the current, so if one trajectory was `A B B B B C D D E F F`, it will be flattened to `A B C D E F` indicating that the user has moved from `A -> B -> C -> D -> E -> F`. This may or may not be the desired result, as one some times only wish to now that a user moved from `A -> F`, but with this method the intermediate buildings also recorded as a building-to-building movement.

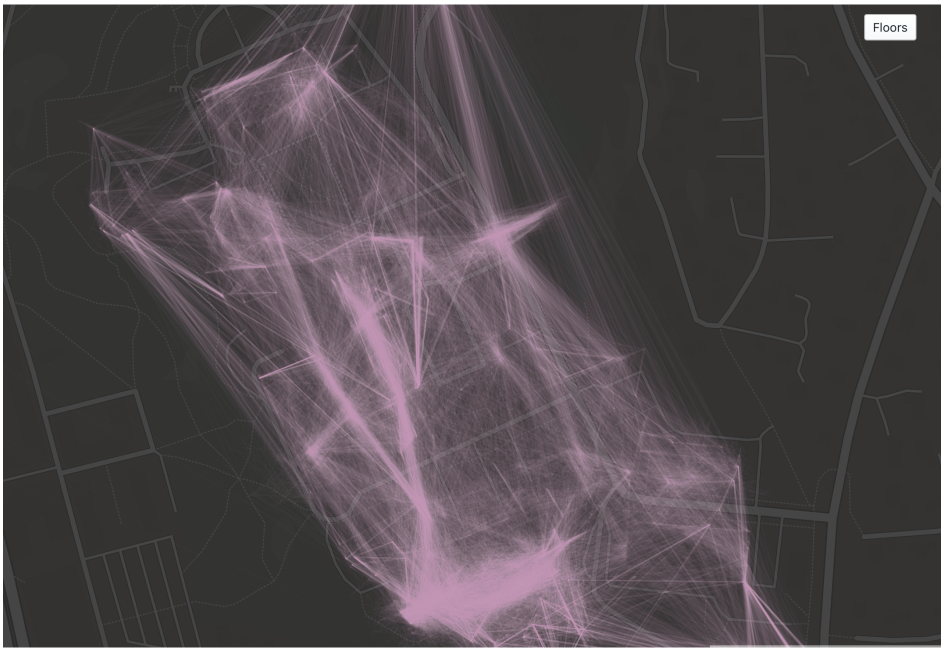
Further, the list is reduced to a building matrix where the number of "transitions" `A -> B` is the value for $BuildingMatrix_{AB}$. The matrix creation could possibly be expressed directly as SQL as well, but it was found to be easier to implement in JavaScript. In the end a GeoJSON feature collection of `LineStrings` is made for each transition, removing those that have a value below a configurable threshold to reduce noise in the visualization. For an enhanced visual appearance, the original `LineStrings` of two points are converted to subtle bezier curves [43] containing many points at the cost of a slight increase in file size.

5.4.4 Mapbox GL JS as a React Component

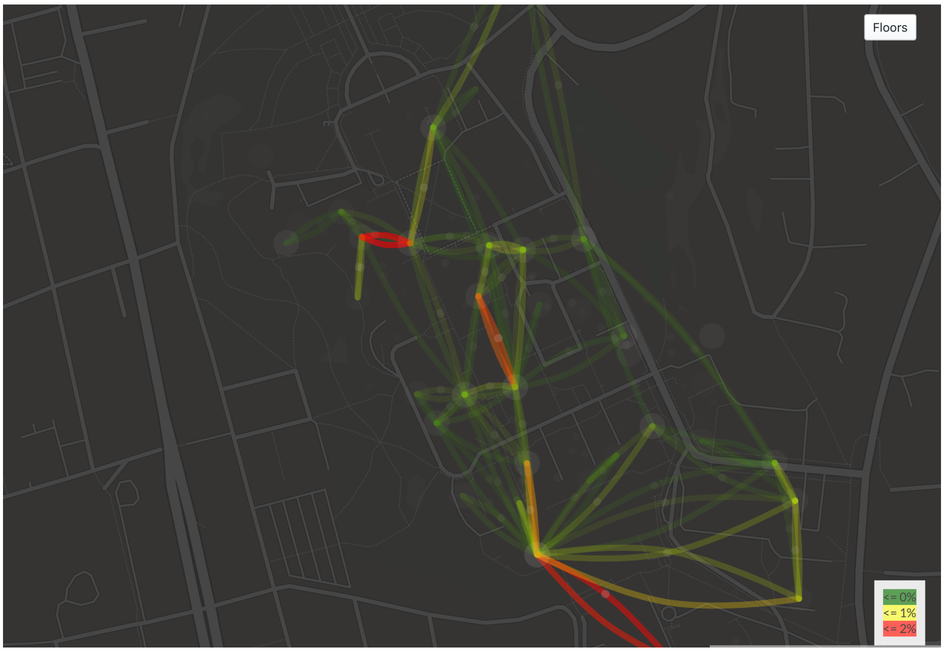
React utilizes a component based approach to user interface development where each component is an independent highly reusable piece. Components are defined as functions returning a representation of *what* the component should look like given a set of properties, and React will go to great lengths to try and determine *how* to do it using various mechanisms such as diff-ing the output against an internal abstract representation of the DOM before calling the real DOM altering methods[2]. This approach ensures that only a minimal amount of changes must performed on the DOM on each update, but does not comply well with libraries written outside this paradigm. As a mean to interface with such libraries, React provides life cycle methods for components that is invoked at certain times during the life cycle of a component, for example when a component is mounted or is going to be unmounted. Mapbox GL JS works on the DOM, so to make it play nicely with React, a higher-order component called `createMapboxContainer` was developed to let the map be controlled using the same standard React and Redux mechanisms used by other parts of the UI, and making sure that the map is ready before attempting to paint on it.



(a) Snapshot



(b) Movement



(c) Buildings

Figure 5.6: The different visualization modes

More screenshots of the application, including the different modes can be found in Appendix A

Chapter 6

Results

This chapter presents results from using the application in form of patterns and phenomena that can be discovered using it. A review of insights from using the application the main research contribution. In addition, it gives an evaluation of its performance characteristic and the capabilities of the system in its current form.

6.1 Patterns and Phenomena

This section illustrates some peculiar patterns and phenomena found by looking at the visualizations using the application developed in this project. These images can shed light on possible errors in the dataset due to inaccuracies, faulty trackers and other unknown reasons. Some of the patterns have been evident also in previous work, but the new visualizations provide a different, yet complementing view.

Outside Buildings

Figure 6.2 indicates that there are a lot of movements west of IT Vest. However possible, it is unlikely that such a large amount of movement should happen there and down the hill right outside. Figure 6.1 reveals a rather impossible movement pattern. It indicates that devices have been positioned outside the building in the upper floors of the tall Sentralbygg 1, which is not likely due to the nature of physics. Using the distinguish floor feature of the application, it is straightforward to conclude which floors are

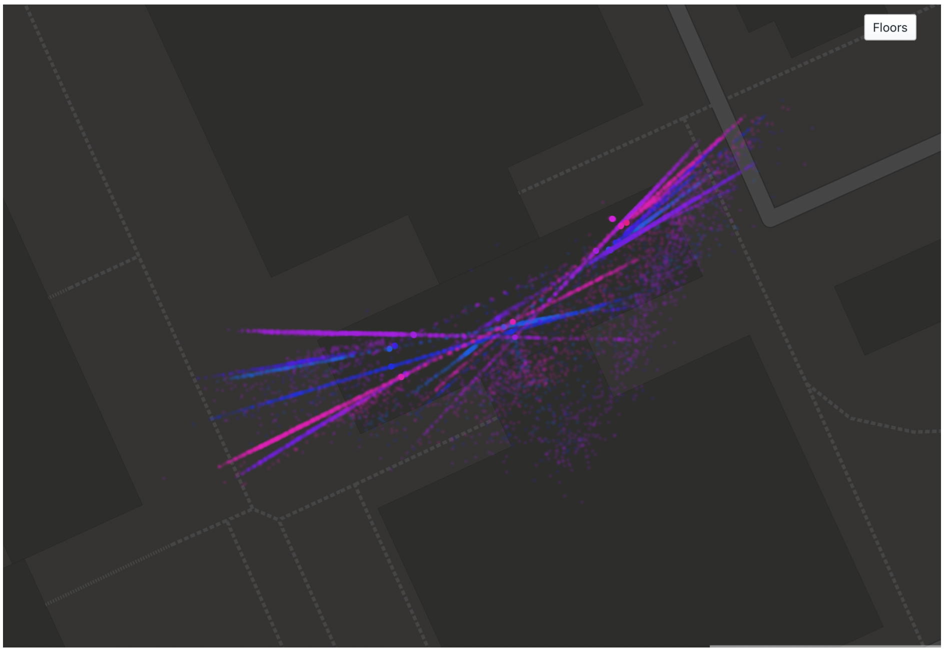


Figure 6.1: Devices floating in the air outside building boundaries

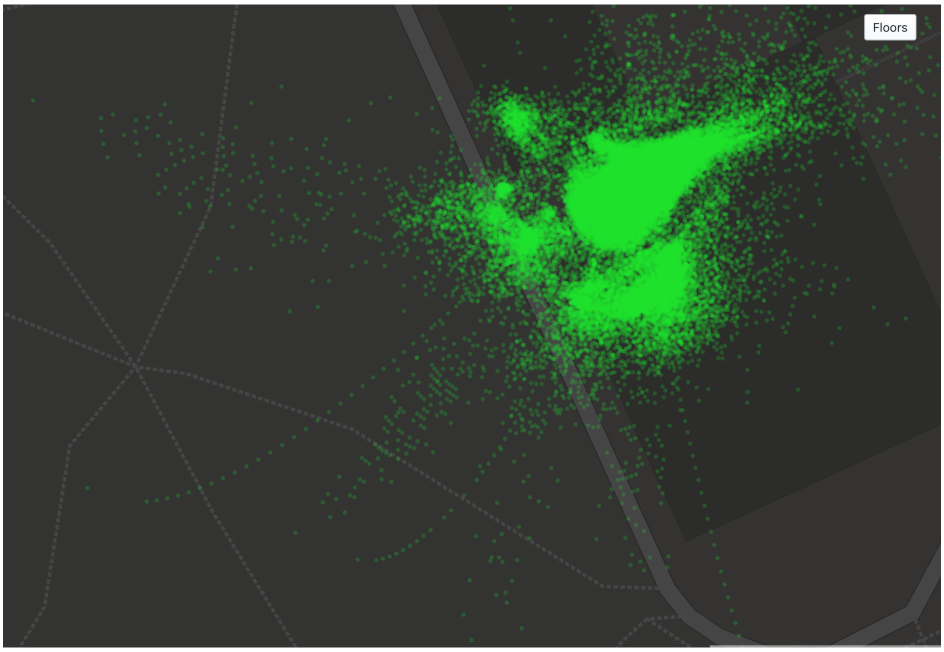


Figure 6.2: Considerable amount of devices outside F1

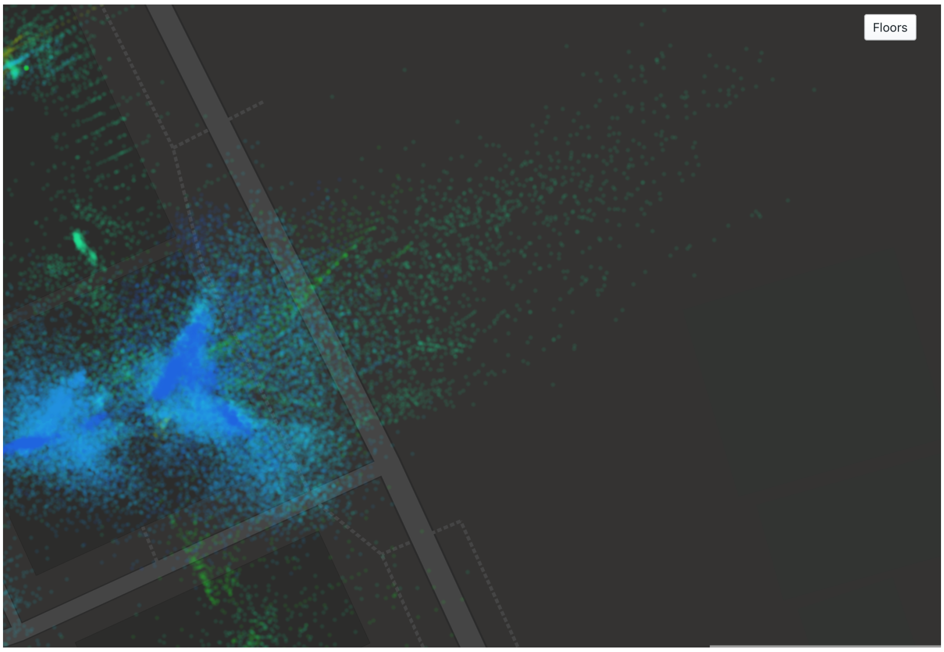


Figure 6.3: Considerable amount of devices outside P15

affected. Figure 6.3 reveals that may be people standing at the bus stop is believed by the tracking system to be on the second or third floor. It also shows many points located in the slopes down to Dødens Dal.

Circular Grids

The patterns illustrated in Figure 6.4 and Figure 6.5 reveal some form of banding where positions are snapped to a circular grid. It is hard to determine the cause of this phenomena, but it can probably be attributed to either the precision of the tracking system or the renderer as discussed in Section 2.5.

Straight Lines

Straight line artifacts are visible throughout the entire campus at any given time, indicating that the tracking system have some issues with regards to pin-pointing exact locations. These were seen in the previous work as well, and the most likely cause is that the access points are positioned in a way that makes triangulation and trilateration inaccurate. Figure 6.6 illustrates this specifically, but the lines can also be seen in all the other figures.

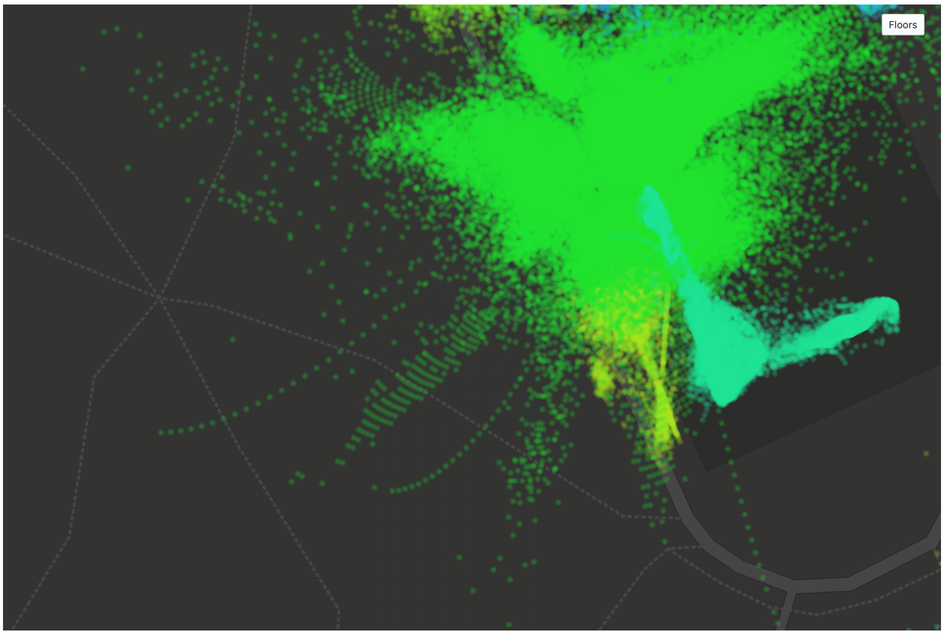


Figure 6.4: Dots in a circular grid

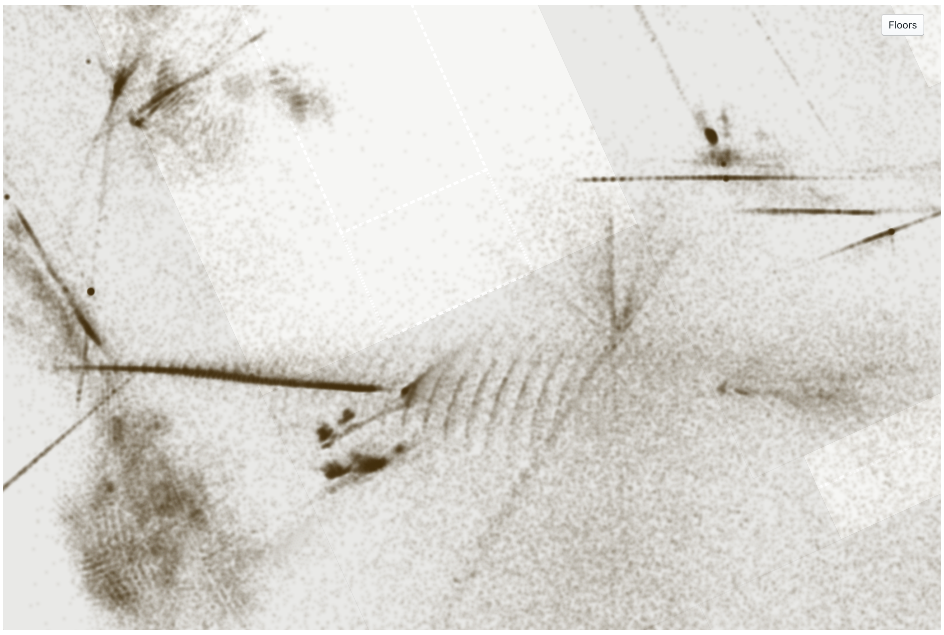


Figure 6.5: Circular grids in the entrance to Realfagbygget

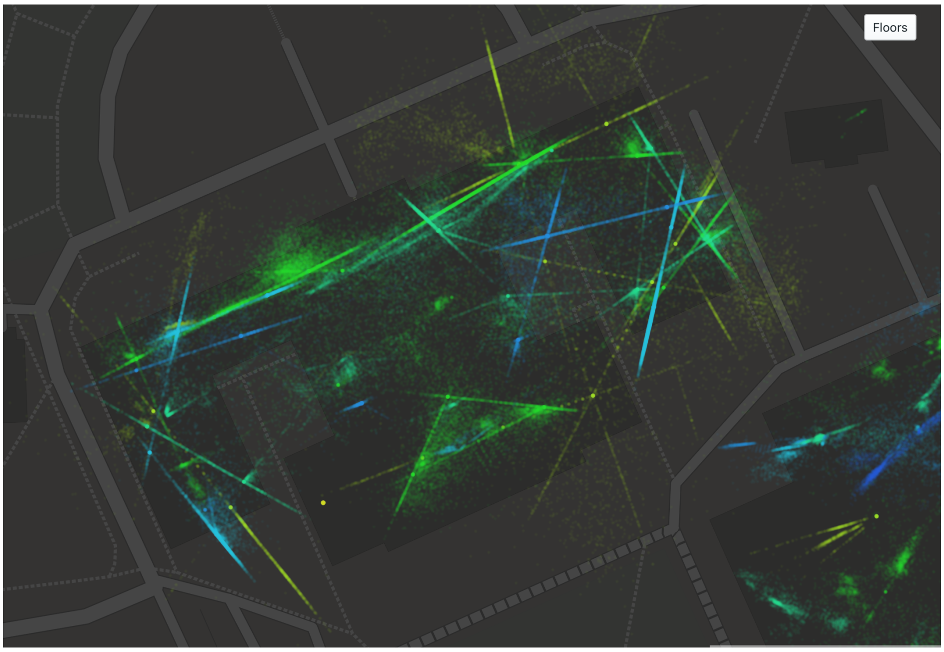


Figure 6.6: Straight Lines are common in the dataset

Inaccurate Trajectories

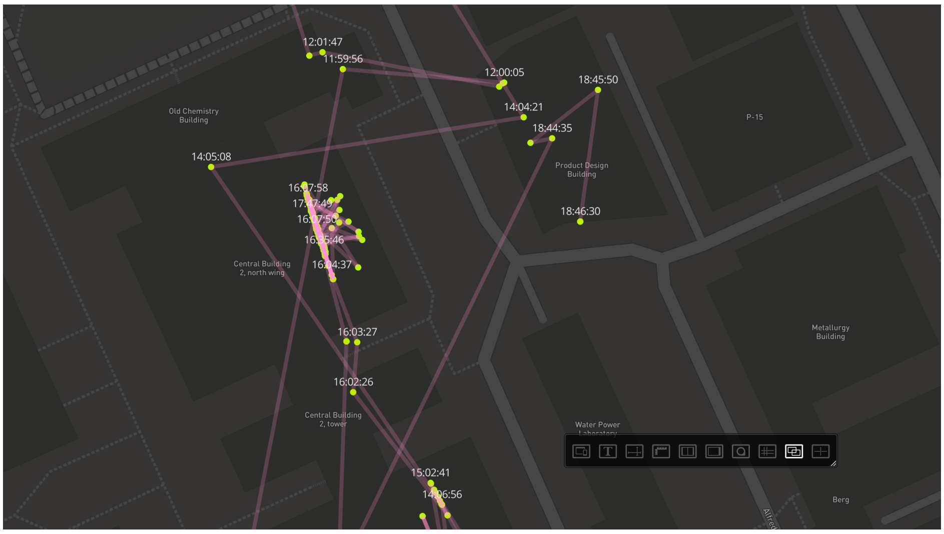
Figure 6.7 shows the trajectory of a single device. There are some clear fluctuations in the trace suggesting that the device is moving back and forth, although it probably stands still. The processed version have filtered out those segments of the trajectory where the speed from the previous was less than 0.2 m/s and distance less than 20 meters, which can be a possible heuristic to smoothen paths.

Uncovered Areas

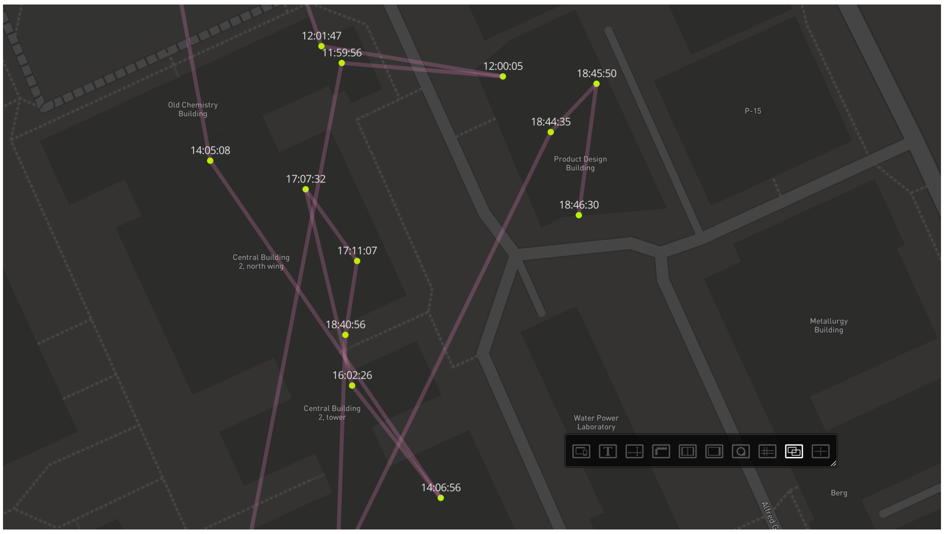
Figure 6.8 illustrates a known fact regarding the tracking system. The SINTEF buildings does not show any recorded positions, as they have their own WLAN system separate to that of NTNU.

Areas No Longer Covered

Figure 6.9 reveals that the tracking may have stopped working in one building at the campus after October 7th 2014. All the days leading up to this date have numerous recordings for this particular building, but after



(a) Original



(b) Processed

Figure 6.7: Inaccurate Trajectories

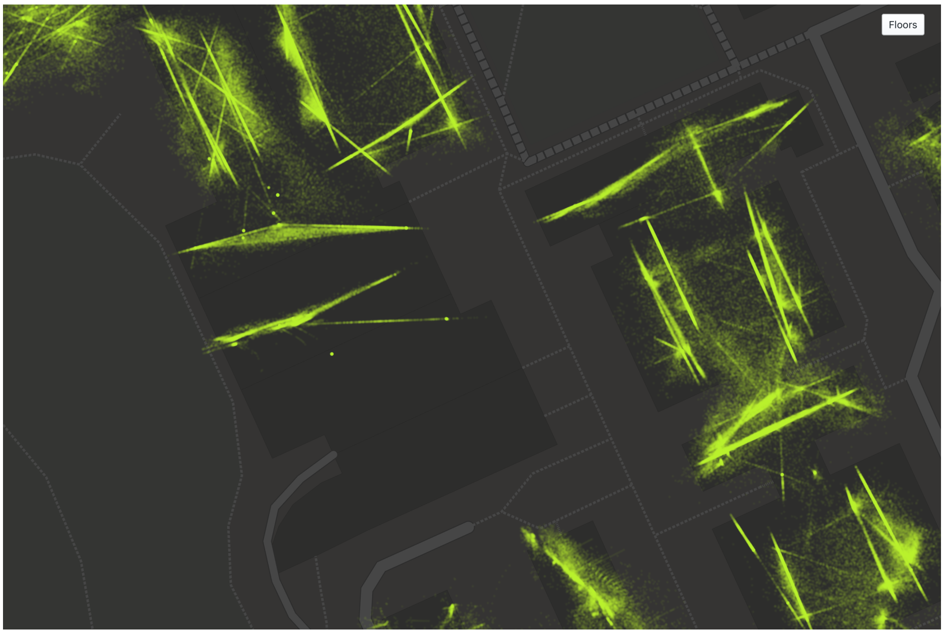


Figure 6.8: Known area not covered

this date they vanish. It shows how an application like this can be use to detect anomalies and broken equipment.

Outliers

By using the Trajectory/Movement view as in Figure 6.10, it can be revealed that there exists extreme outlier records. From the image one can spot trajectories that goes through points far north-west of campus. This must be an error since the recordings in this dataset should only be from the NTNU Gløshaugen campus.

6.2 Performance Measurements

The most important aspects in terms of performance when it comes to an web application like **Crowds**, from a user's point of view, is the time taken to find and process the data from the storage layer, as well as the time and resources required to transfer the tiles from the server and actually *render* and interact with the visualization on the client. The former is mostly limited by the available computing power on the server and how the database is indexed and the processing algorithms, while the latter is limited by factors such as network and available hardware on the user's computer, which is less easily controlled than the server resources. The hardware on the server can easily be managed and changed to accommodate new requirements by the people responsible for technical matters in an organization given enough funds. In this section, performance characteristics of the application will be evaluated.

The server component of the system was running on a desktop machine with an Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz with 16 GB of RAM and a SSD drive. The client was tested on a 2016 MacBook Pro with a 2.0GHz dual-core Intel(R) Core(TM) i5 CPU and a Intel(R) Iris Graphics 540 GPU.

6.2.1 Fetching Data From the Database

All of the tests were performed in serial to avoid the penalty each test could receive by running them in parallel which could invalidate comparisons.

Figure 6.11 shows that the time taken to execute a range snapshot query scales linearly with the number of rows retrieved. Each line in the graph represents a query for one particular day of a week run with different range sizes yielding a different row count. The queries were run multiple times and the median of the execution time was plotted. The execution times were captured using the `EXPLAIN ANALYZE` feature of PostgreSQL. By prepending this statement to a SQL query, the query will be planned and actually executed returning statistics such as elapsed time and row count. However, it does not return the rows that were found to the client. Due to this fact, the elapsed time do not include the transfer time of data from disk, but this can be calculated given the specification of the disk drives installed.

The dataset used in this project can not really be classified as Big Data. None of the three Vs of big data is fulfilled. The *volume* can seem large, but it can easily fit into RAM of a well-equipped modern server as it is less than 100GB. The data is historical and static, so there is no *velocity*. Finally, all the records have the same structure yielding no *variety*. However, once data is collected in real-time at regular intervals, these characteristic will be met and one can talk about it as big data [24].

Consequently, the entire dataset could fit into RAM of a decently equipped server, which should in theory yield a speedup of orders of magnitude. This is a question of cost / utility, but RAM is not to expensive, and if one were to do some real analysis it may be worth it.

Timings for three different phases as a function of time range sizes are shown in Figure 6.12, Figure 6.13 and Figure 6.14 for snapshots, movements and buildings respectively. The **Explain Analyze** shows the time taken to find the correct records minus transfer time from disk, **pg2geojson** shows the time taken to produce the GeoJSON, and **geojson2mbtiles** the time taken to build the MBTiles from the GeoJSON. The graph shows that it scales linearly with the range size (which should correspond to increased row count). Figure 6.13 show that the data processing stops for ranges over 48 hours because resources have been exhausted. The **explain analyze** in the database continues confirming that the bottleneck is the processing code.

Figure 6.17 shows how the size of the MBTiles and GeoJSON evolve for each of the modes as a function of the data size. For snapshots both increase linearly with different slopes, whereas the MBTiles are nearly constant for buildings. Buildings have a theoretical maximum of $N \times N$ lines where N is the number of buildings, so we see that the curve flats out as expected. Most interesting is probably the results for movements. It shows that the MBTiles get exceedingly large compared to GeoJSON for on-disk storage. Further, one can see that the graph stops at about at 110,000 devices meaning that processing failed when data reached this threshold. It is suspected that the processing code runs out of RAM. Given the large MBTiles size for movement trajectories, it might be reconsidered to use this for this mode due to its large server resource consumption. However, the transfer speed benefits to the client is still present.

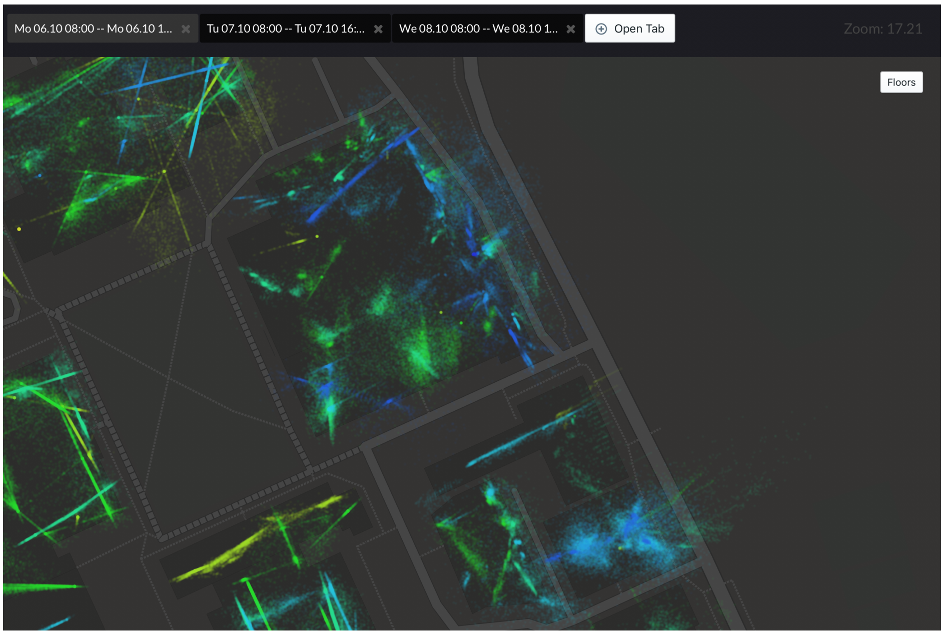
6.2.2 Tile Optimizations

Tippecanoe can optimize the generated tiles by simplifying the geometries at different zoom levels. Figure 6.15 illustrates how using these optimizations affect the visual outcome. The parameter `--base-zoom` defines the level at or above which all points are intact, so by setting this low, most of the tiles will have all data. It is worth noticing that the small geographical area in question here is only visible at high zoom levels. The options `no-skip-tiles` and `no-skip-features` can be set to disable optimizations altogether.

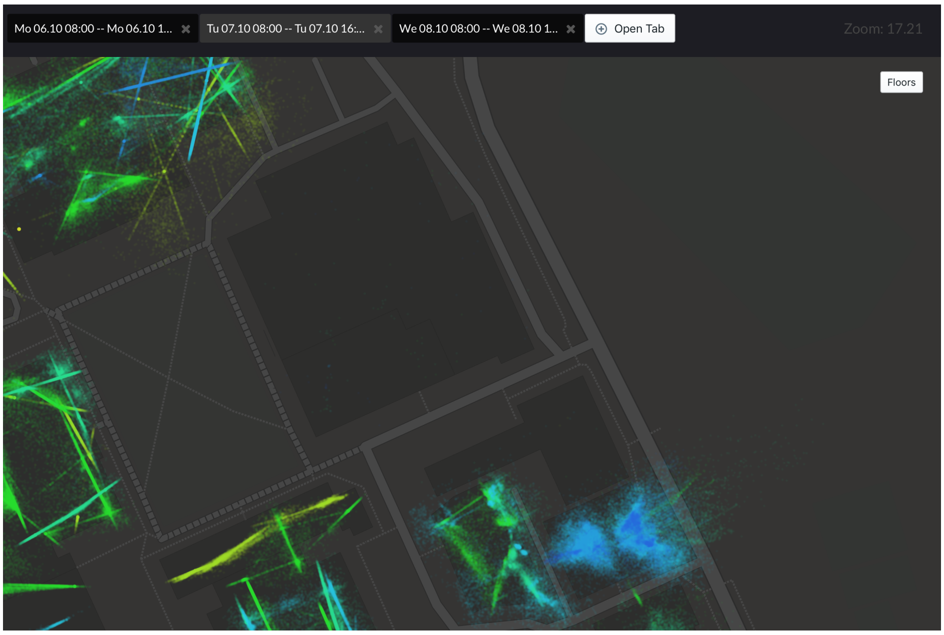
Enabling the optimizations make the tile sets smaller in physical size by *removing* data. This can be acceptable at the lower zoom levels as long as you can get the full and authentic picture when zooming in. At lower zoom levels, geometries tend to have a higher degree of overlap, possibly making it visually indistinguishable. Also, it is not always desired or even possible to get the full detail at lower zoom levels. However, these kinds of optimizations is likely more useful when a larger geographical area is studied.

The experimentation shows that the disk size of one MBTile is roughly the same as the size of its gzipped GeoJSON counterpart, but approximately 30% of the uncompressed GeoJSON. Clearly, for disk storage and network transfer the compressed version is the most important. This result shows that there are no savings of storing the MBTiles vs GeoJSON regarding server disk space when the GeoJSON is stored gzipped. However, when transferring data to the browser there are considerable savings. Recall from Section 2.8.3 that the server need not send all the tile data, only what is currently in the viewport at a given zoom level. Thus, the net amount of transferred data dramatically decreases when using this approach. Granted, if the user interacts with the map in a way that the entire map becomes visible, and thus incurs loading lots of tiles, the total amount transferred may exceed that of loading the GeoJSON beforehand, but the perceived performance was increased by having visualizations rendered progressively.

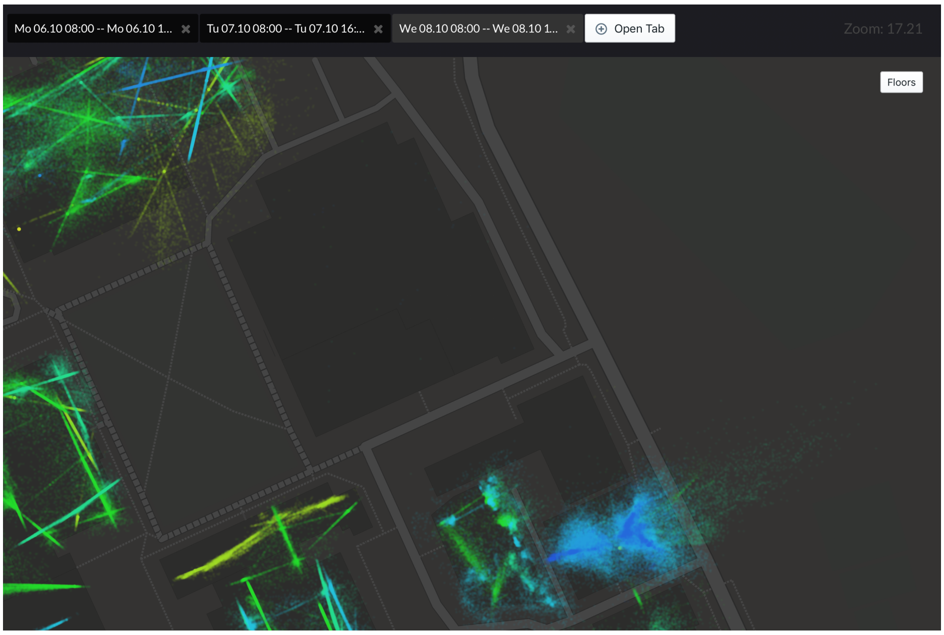
For example, a tile set containing 8 hours of data between October 1st 2014 at 8:00 and October 1st 2014 16:00, the gzip compressed GeoJSON has a file size 17 MB, the entire set of MBTiles is also 17 MB, but the client only needs to download 12 tiles (Figure 6.16) with a total size of 2.73 MB to render the tiles needed to cover the viewport at zoom level 16.


Mo 06.10 08:00 -- Mo 06.10 1...
Tu 07.10 08:00 -- Tu 07.10 16:...
We 08.10 08:00 -- We 08.10 1...
Open Tab
Zoom: 17.21
Floors

(a) Monday 6th of October


Mo 06.10 08:00 -- Mo 06.10 1...
Tu 07.10 08:00 -- Tu 07.10 16:...
We 08.10 08:00 -- We 08.10 1...
Open Tab
Zoom: 17.21
Floors

(b) Tuesday 7th of October


Mo 06.10 08:00 -- Mo 06.10 1...
Tu 07.10 08:00 -- Tu 07.10 16:...
We 08.10 08:00 -- We 08.10 1...
Open Tab
Zoom: 17.21
Floors

(c) Wednesday 8th of October

Figure 6.9: Building suddenly becoming uncovered

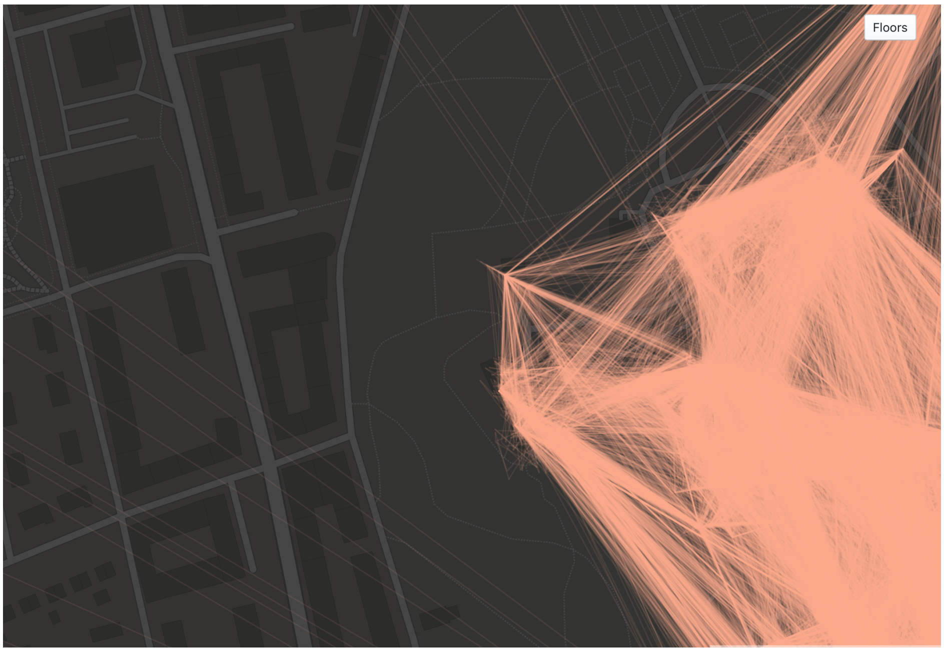


Figure 6.10: Records that are clearly outliers

Note the trajectories heading north-west

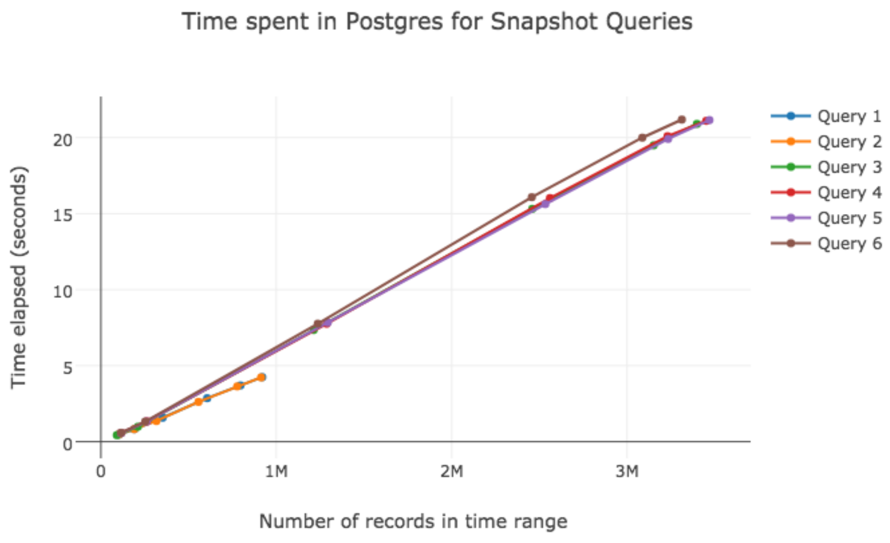


Figure 6.11: Query Time

Hours	**Rows**	**Analyze (s)**	**JSON (s)**	**MBTile (s)**	**JSON (MB)**	**MBTile (MB)**
2	46,282	0.25	2.83	2.02	11.81	4.17
4	89,895	0.45	2.77	2.00	11.81	4.16
6	133,419	0.66	5.11	3.44	20.65	6.93
8	212,442	1.00	6.50	4.41	25.89	8.54
10	585,744	2.75	7.15	4.79	28.25	9.24
12	1,089,626	5.28	7.48	4.99	29.39	9.58
14	1,504,540	7.98	7.73	5.02	30.18	9.81
16	1,504,592	10.04	8.08	5.13	30.87	10.01
18	1,856,085	11.84	8.31	5.21	31.54	10.20
20	2,092,119	12.88	8.67	5.49	32.94	10.62
22	2,223,229	13.41	11.08	6.96	41.68	13.30
24	2,296,170	13.72	14.45	9.09	54.19	17.10
26	2,350,931	13.94	18.32	11.34	68.14	21.21
28	2,401,834	14.21	22.12	13.62	81.16	25.02
30	2,452,354	14.44	24.76	14.99	89.66	27.53
32	2,536,820	14.84	26.08	15.88	94.19	28.87
34	2,900,398	16.60	26.92	16.00	96.52	29.57
36	3,414,160	19.18	27.15	16.32	97.72	29.93
38	3,974,005	22.11	27.57	16.59	98.54	30.16
40	4,508,362	24.89	27.67	16.76	98.71	30.21
42	4,879,315	26.84	28.55	16.75	99.36	30.40
44	5,110,501	27.95	28.65	17.07	100.71	30.80
46	5,248,653	28.49	31.54	18.39	108.94	33.23
48	5,332,929	28.84	34.56	20.42	120.89	36.75
56	6,998,437	42.50	46.06	26.42	157.71	47.53
64	7,344,297	44.26	47.75	27.56	162.97	49.07
72	8,320,520	50.38	54.17	30.58	183.37	54.95
80	9,953,053	60.93	65.06	37.71	219.45	65.23
88	10,265,007	62.27	68.97	38.46	223.83	66.50
96	11,126,514	67.64	72.74	43.41	240.98	71.41

Table 6.1: Tile Data for Snapshot

Hours	**Rows**	**Analyze (s)**	**JSON (s)**	**MBTile (s)**	**JSON (MB)**	**MBTile (MB)**
2	46,282	0.24	2.67	0.82	8.71	0.39
4	89,895	0.45	2.60	0.80	8.71	0.38
6	133,419	0.67	4.77	1.46	15.84	0.70
8	212,442	0.99	6.16	1.80	19.74	0.90
10	585,744	2.27	6.80	2.00	21.99	0.99
12	1,089,626	4.12	7.17	2.06	22.67	1.03
14	1,504,540	6.07	7.49	2.10	22.82	1.04
16	1,504,592	10.09	7.77	2.10	23.05	1.06
18	1,856,085	12.48	8.04	2.11	23.05	1.05
20	2,092,119	13.58	8.52	2.17	23.87	1.07
22	2,223,229	14.27	14.46	2.42	26.42	1.19
24	2,296,170	14.65	17.82	2.66	29.05	1.28
26	2,350,931	14.98	21.49	2.92	31.83	1.35
28	2,401,834	15.30	25.36	3.21	35.06	1.44
30	2,452,354	15.64	28.36	3.42	36.78	1.51
32	2,536,820	16.09	29.97	3.45	37.53	1.55
34	2,900,398	17.80	30.81	3.49	38.06	1.57
36	3,414,160	21.28	31.61	3.54	38.36	1.58
38	3,974,005	24.25	32.10	3.54	38.36	1.59
40	4,508,362	27.17	32.54	3.54	38.36	1.58
42	4,879,315	29.28	32.76	3.54	38.36	1.58
44	5,110,501	31.39	33.20	3.56	38.81	1.60
46	5,248,653	31.35	35.48	3.75	40.38	1.65
48	5,332,929	31.83	39.01	3.86	42.04	1.70
56	6,998,437	46.88	51.67	4.41	47.74	1.94
64	7,344,297	49.48	54.07	4.45	48.04	1.96
72	8,320,520	55.84	61.14	4.97	50.75	2.09
80	9,953,053	68.82	74.89	5.13	54.95	2.24
88	10,265,007	70.41	77.06	5.17	55.25	2.26
96	11,126,514	76.20	82.37	5.24	56.15	2.30

Table 6.2: Tile Data for Buildings

Hours	Rows	Analyze (s)	JSON (s)	MBTile (s)	JSON (MB)	MBTile (MB)
2	1,858	0.14	2.38	32.70	3.59	124.91
4	2,050	0.26	2.32	31.97	3.59	124.94
6	2,267	0.39	4.39	33.37	5.47	134.12
8	6,044	0.58	5.78	33.79	6.50	139.20
10	16,646	1.36	6.37	33.94	7.01	141.99
12	24,290	2.45	6.74	33.82	7.27	143.28
14	29,616	4.98	7.25	37.95	7.49	158.27
16	29,616	6.85	7.55	37.89	7.60	158.51
18	33,954	8.65	7.61	37.84	7.68	158.74
20	36,208	9.42	8.25	40.09	8.31	166.76
22	37,371	9.89	14.45	100.88	10.83	394.82
24	37,901	10.15	17.69	168.33	13.20	640.94
26	38,954	10.38	21.90	202.46	15.55	775.49
28	39,155	10.55	25.99	217.04	17.67	858.58
30	39,328	10.85	29.23	218.63	19.28	866.46
32	43,024	11.19	31.31	219.86	20.28	871.39
34	54,042	12.49	32.40	221.25	20.81	887.76
36	61,254	15.10	33.00	223.41	21.09	896.67
38	67,242	17.18	33.48	228.62	21.32	918.37
40	72,190	19.19	34.56	230.41	21.35	918.53
42	75,795	20.63	35.16	230.59	21.45	918.82
44	77,881	21.61	36.77	247.67	22.12	985.89
46	79,064	22.20	38.67	293.52	24.65	1180.03
48	79,601	22.51	43.07	335.76	27.14	1355.84

Table 6.3: Tile Data for Trajectory. The row counts are smaller than in snapshot and buildings due to the rows being grouped by device.

Option	Description
no-skip-features	Don't skip features
no-skip-tiles	Don't skip tiles
base-zoom	16

Table 6.4: Tippecanoe Options

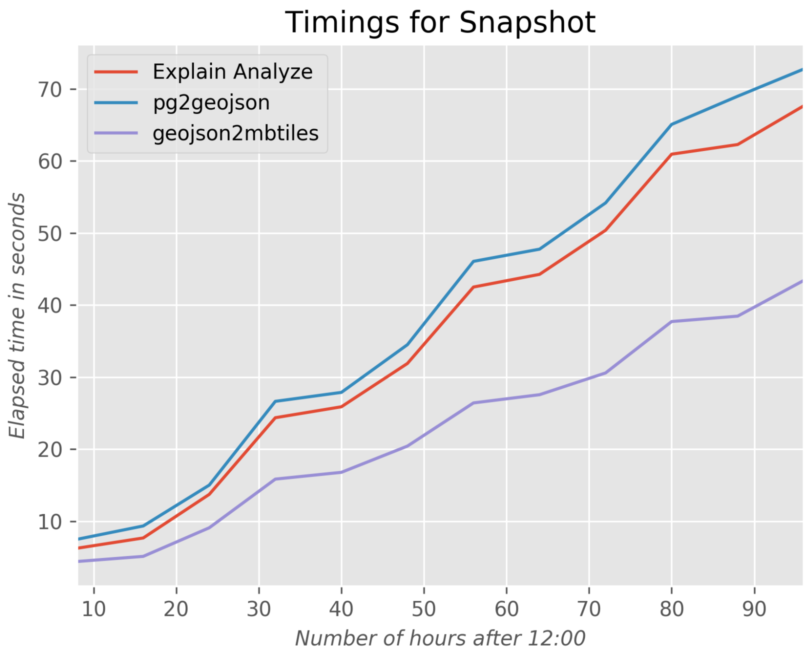


Figure 6.12: Timings for queries run in *Snapshot* mode.

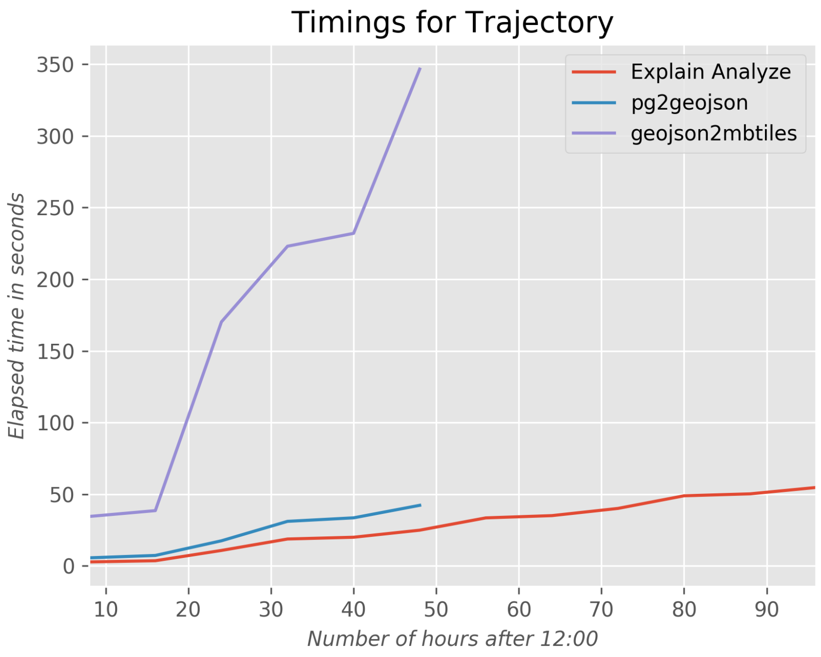


Figure 6.13: Timings for queries run in *Movement* mode.

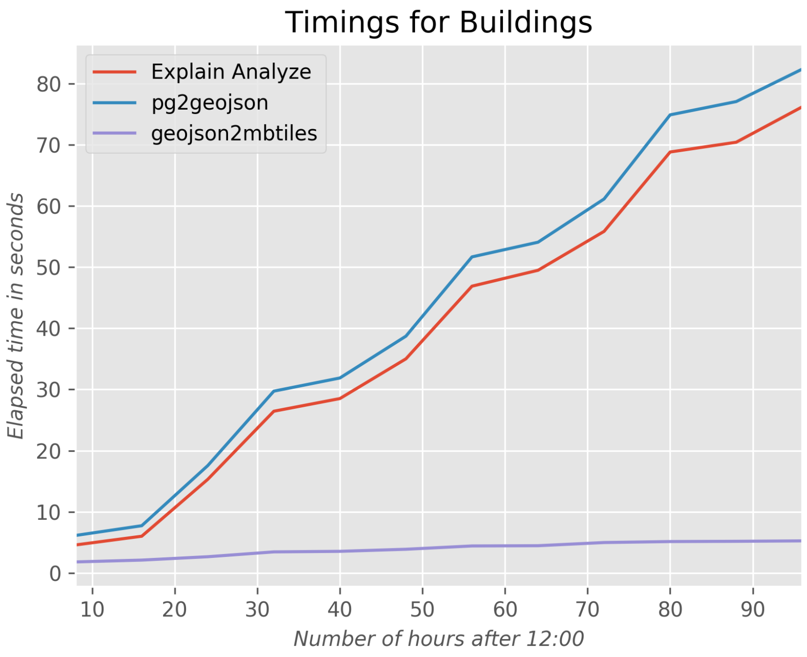
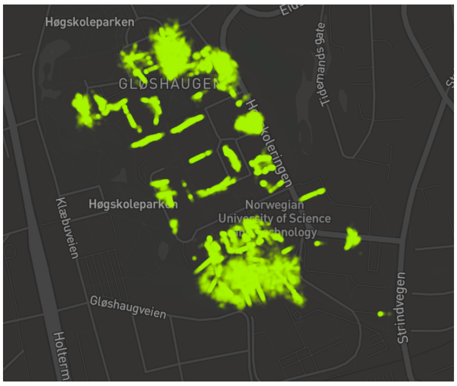
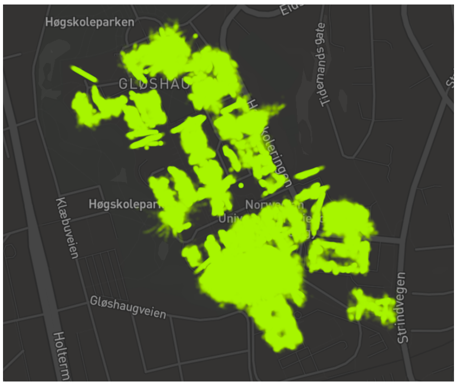


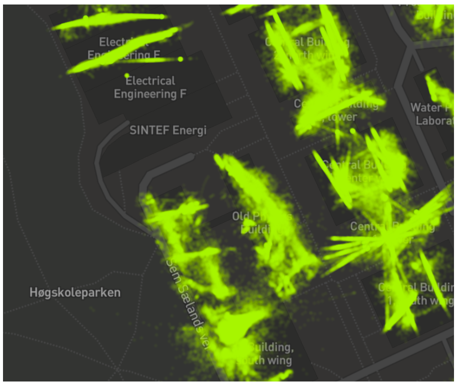
Figure 6.14: Timings for queries run in *Buildings* mode.



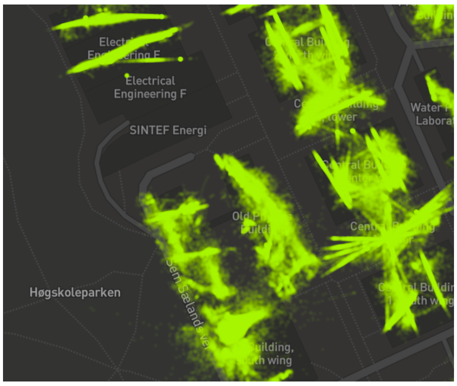
(a) baseZoom=16 at z=14



(b) baseZoom=14 at z=14

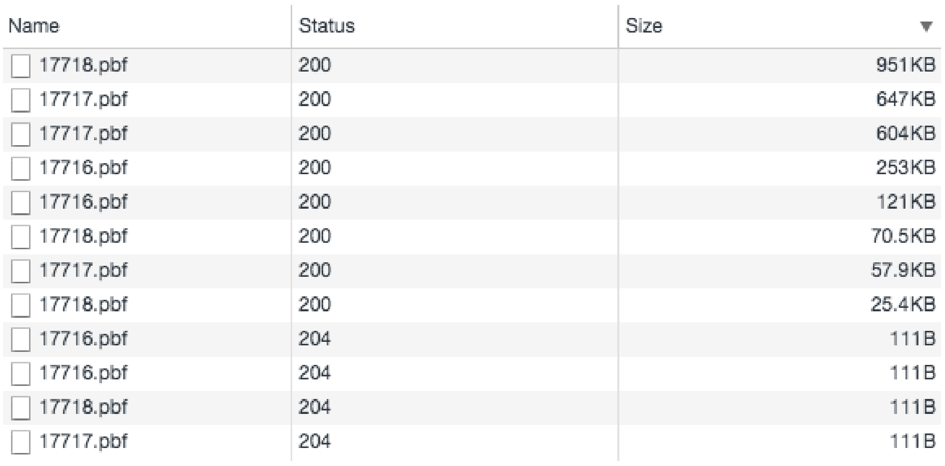


(c) baseZoom=16 at z=16



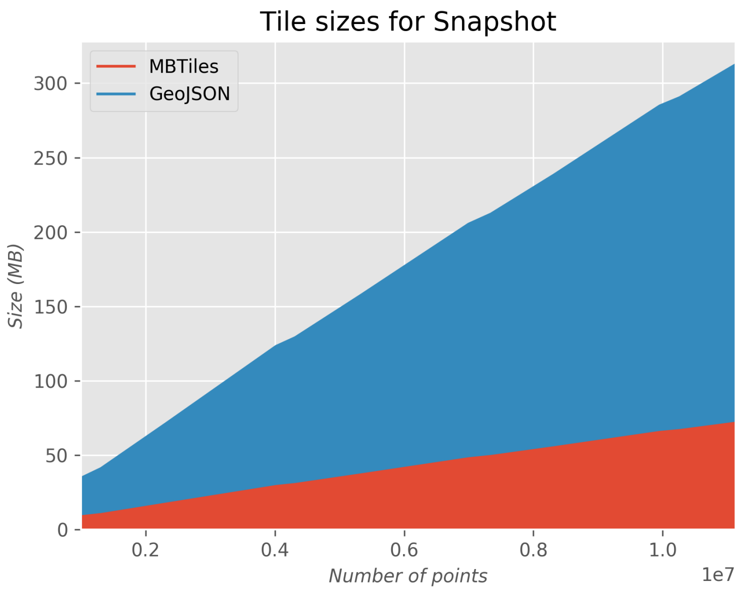
(d) baseZoom=14 z=16

Figure 6.15: Effects of using Tippecanoe Optimizations

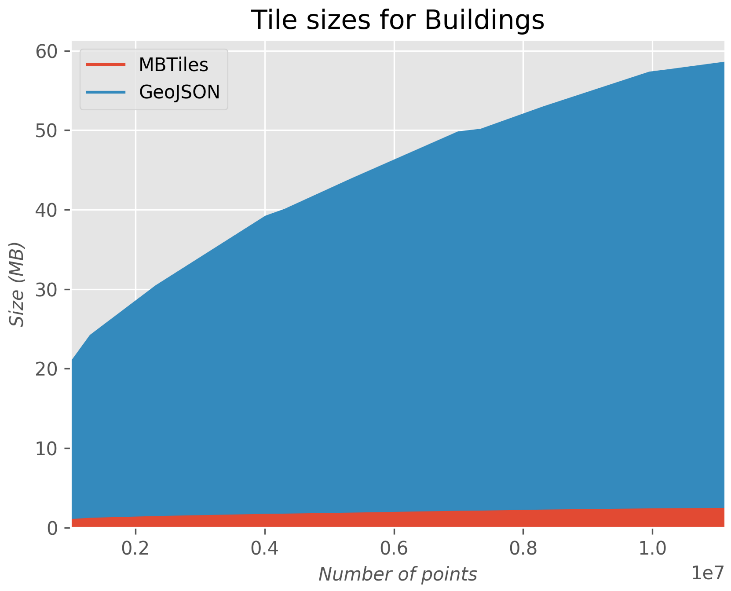


Name	Status	Size ▼
17718.pbf	200	951KB
17717.pbf	200	647KB
17717.pbf	200	604KB
17716.pbf	200	253KB
17716.pbf	200	121KB
17718.pbf	200	70.5KB
17717.pbf	200	57.9KB
17718.pbf	200	25.4KB
17716.pbf	204	111B
17716.pbf	204	111B
17718.pbf	204	111B
17717.pbf	204	111B

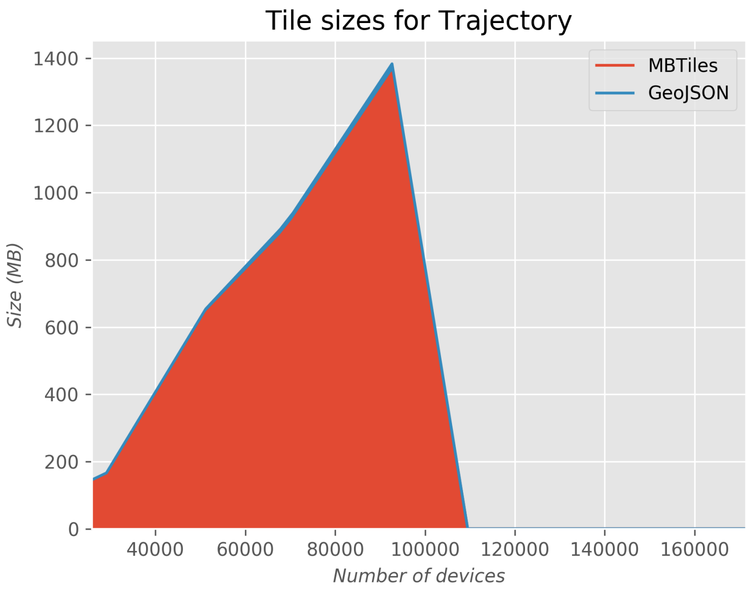
Figure 6.16: Screenshot of requests made by Chrome for zoom level 16



(a) Snapshot



(b) Buildings



(c) Movement

Figure 6.17: File size of MBTiles and GeoJSON

Chapter 7

Discussion

7.1 Patterns

The patterns visualized using the application has indicated that there are inaccuracies in the position data, and different phenomena that occurs for currently unknown reasons. There is no way to be absolutely certain of reasons for these phenomena. For device trajectories possible reasons for seemingly wrong or unusual patterns may include factors such as

- Device batteries lost: Devices may run out of batteries at some time during a movement.
- People making unexpected turns due to some external factor such as seeing or meeting acquaintances
- Walking at the edge of coverage. People may be wandering in areas were there are little or no coverage, e.g. between the reach of two access points. When people are moving between two buildings, the position may be assigned to either of the buildings and even be assigned to both in a single movement.
- People turning off Wi-Fi on their devices for battery saving purposes.
- People wanting to explore new paths on campus different from what is considered the "best" path.

These also apply in the general case, along with the possibility that there is an inherent inaccuracy in the positioning hardware or that its it miscalculating. It is not easy to determine the root cause of these patterns, and some patterns may be unusual, yet in fact real. The equipment do not have extreme accuracy (around 10 meters), and to figure out to which degree a pattern is wrong or right, one must compare with data from different equipment to increase the certainty of either possibility. One could imagine trying to apply machine learning on the dataset to determine correct positions, but it would need correct training data impossible to get if one does not have a perfect tracking system. The patterns found using the application can aid in future development of cleaning data cleaning methods, and the application itself can be used to test such methods.

7.2 Performance

The perceived performance is the performance experienced by the user interacting with the systems. By focusing on perceived performance one can increase a users' productivity [42]. Having fluid transitions and responsive user interface elements are vital in this manner. Some metrics to measure perceived performance include:

- **First Paint** The first time something actually becomes visible on screen. It does not imply that everything is available, only that *something* is present. In this project, it is not really an interesting metric apart from confirming that loading is actually in progress and the server is not down.
- **Time to Interactive** The time taken from opening a web page until the user can interact with it. A very important metric for applications relying heavily on user interaction, but less important for read-only content.
- **Frames Per Seconds** Measure the number of frames rendered by second, 60 FPS is considered a requirement for a 100% fluid experience in games and animations, but less FPS may be acceptable in selected cases.

An important factor for great perceived performance is that the UI thread is not blocked. Traditionally, heavy CPU computations on the main browser thread has penalized performance and rendered apps virtually useless during these computations due to an unresponsive interface. However, with the advent of Web Workers, processor intensive work can be run in background threads, leaving the UI responsive and open to user interaction. In this application, computing map projections are done in Web Workers and rendering is done by the GPU, so the user interface is not blocked during these operations. Users can therefore continue using the application while maps are rendering.

When the time range exceeds 48 hours, the panning interaction begins to feel sluggish on the MacBook and the perceived performance decreased. However, it is still to be considered usable. This range corresponds to over 5 million points as shown in Table 6.1. 72 hours worth of data can also be rendered, albeit with even slower interactions. For data corresponding to time ranges above this the performance starts to suffer and loading times increase due to large tile sizes.

7.3 MBTiles vs GeoJSON

The results has shown that using the tiling approach facilitated by MBTiles can give a gain compared to sending raw GeoJSON. The gain will become even more prominent when the geographical area in question is larger, and thus less of the entire space is visible in the viewport at once. Using MBTiles, smaller chunks of data can be loaded progressively and rendering can start immediately. With raw JSON on the other hand, everything must be loaded upfront [1]. Although, in the end the total amount of data to be rendered will be the same, resulting in the same limits for how much data the rendering engine can practically deal with. The experiments also revealed that the MBTiles got exceedingly large for line trajectories, and that for this particular application, they may not be the best fit despite their benefits when it comes to loading features onto the client map.

In any event, the research shows that it would require large amount of disk space to run this application. Not only is the database itself rather large, but with all the cached tiles that accumulate after using the application for a period of time, the disk requirements will continue to increase. But in the current iteration of the application, it was found necessary as it would take too long time to wait for the database on every request for a particular slice of the dataset.

7.4 Fulfillment of Requirements

Section 4.3 defined a set of requirements for the application. Table 7.1 shows to which degree these were considered fulfilled.

Requirement	**Fulfilled**	**Comment**
FR1	Yes	
FR2	Yes	Everything can be explored, but not at once
FR3	Yes	There are limits to how large the ranges can be
FR4	Yes	
FR5	Yes	Object of interests currently include only buildings
FR6	Yes	
FR7	No	
FR8	Yes	
FR9	Partially	Only for the buildings mode
NRF1	Yes	
NRF2	Partially	Depends on the system to integrate with
NRF3	Yes	Standards have been used where applicable

Table 7.1: Fulfillment of Requirements

7.5 Usability

A thorough usability test on different user groups has not been carried out in this project, as it was suspected that the feedback would most likely coincide with data gathered in previous work. The application in its current state is more of a research platform, than tailored to a specific business case. The usability is therefore not measured in terms of factors such as *perceived usefulness* or *perceived ease of use* [16], but rather in terms of achievable performance and how it can be used to shed new light on presented data.

Although evaluating the usability using standard methods has not been a primary focus, making the system easy and fairly intuitive to use has been a goal and being iterated on through the lifetime of the project. Initially, the prototype focused more on backend systems with a very basic user interface, but later iterations have been incorporating new usability features suggested by people casually testing the application.

7.6 Integrating With Other Systems

Most of the data used by this platform uses standardized formats such as GeoJSON and Vector Tiles. By using standardized techniques it will most likely be easier to integrate with other platforms, as suggested by the feedback received in Aulie's work[10]. Vector tiles are supported by multiple web map solutions, with increasing support, meaning that it should be possible to interchange the underlying technologies without making already processed data unusable. Since the raw data reside in a database universally supported on many platforms, it is also easy to use the same data in other applications where requirements are different. For example, if a researcher only needs to aggregate numbers of the data, writing a SQL query is straightforward.

7.7 Feedback From Others

The system was presented in a meeting with Morten Hatling from SINTEF and he provided some feedback on the prototype and what kind of tasks it could be used for in selected contexts. In the meeting a few specific areas were concretized. It was mentioned that it would be useful to specialize the application to an even smaller scale. For some stakeholders only one building is of interest, and it could be useful to focus the application around specific buildings. This would in turn decrease the amount of data necessary for a visualization, increasing the performance of the system.

7.8 Aggregate Dates vs. Date Ranges

As discussed in Section 5.3.2, the application currently supports two different modes of slicing data at date boundaries. Range mode selects contiguous data between two given timestamps. Aggregate on the other hand takes an array of days and and array of hours, and uses the `day_of_week` and `hour_of_day` columns to find matching entries. This allows the user to request data from "Mondays at 12" or "Saturdays and Sundays between 10 and 12". These kinds of queries are fairly straightforward to express using SQL. However, data sizes can quickly escalate when using the aggregates, so with the current approach is must be used with care. Ideally, this feature could give the user the possibility of comparing weekdays to weekends and even holidays to regular days, but for the largest of the aggregates this is not really feasible in the current state due to the number of points.

7.9 Other Potential Usages

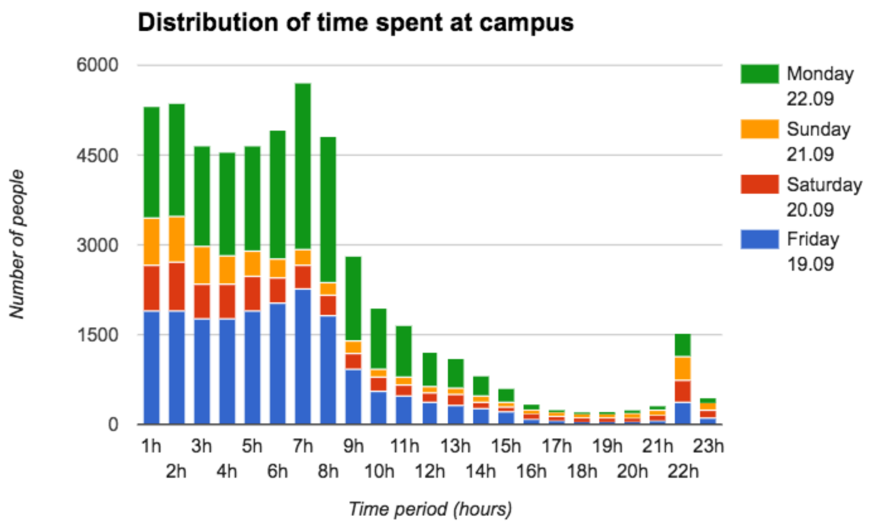


Figure 7.1: Distribution of time spent at campus

Figure 7.1 shows an approximate distribution of how long people stay at campus for four given days. Each bar represents one hour up to the next, meaning that the second bar shows number of people who have been assumed to be at campus between one and two hours, whereas the first shows people who have been there for less than an hour. There are of course some inherent faults with these results. It only measures people whose devices have been recorded more than one time, and will not take into account devices that have lost battery or people who have left and come back one or more times. The actual count could therefore be much less or much larger. Nevertheless, it illustrates a possible use case using an aggregation of tracking data, and also states the usefulness of a structured query language for analyzing data. Having a database with great query capabilities makes it easier to ask different questions about the data than having it in e.g. JSON files, as was done before.

Chapter 8

Conclusion and Future Work

This project has investigated how an improved web application can be developed to help visualizing considerable amounts of positioning data in a performant manner, to aid in the process of discovering or confirming known and unknown patterns in the data. At the same time it has demonstrated the capabilities of the web platform and PostGIS for such applications. The application has been built using popular industry standard tools and is to be considered usable for further research tasks. The results from exploration using the tool provide both new and confirms previously seen patterns in the available dataset, and can be a valuable tool for when reasoning about the dataset and developing data cleaning methods. The performance of the artifact has been evaluated and patterns from the dataset have been presented. The research shows that PostgreSQL with PostGIS and Mapbox provide a good foundation for building position data visualization apps that can be run in a browser. The knowledge contribution from this Design Science Research project (see Section 3) is images of various phenomena in the dataset and an evaluation of the fruitfulness of using web technologies for building the tools to create them.

8.1 Research Questions

RQ1: *What are the limits of the web platform and PostGIS for building data-heavy visualization applications?*

The results show that rendering performance starts to suffer in the browser when the amount of features to be rendered exceeds roughly 8 millions. It was expected that there would be a limit as to how much it could handle, but it nevertheless shows that the browser is capable of handling fairly large quantities. PostGIS was found to be a useful tool when dealing with geographical data, it provides an easy mean of querying the data and can also support more intricate queries than what has been used here.

RQ2: *Is it possible to build a reasonably performant visualization using existing web maps and database technologies?*

The research concludes that it is possible to build reasonably performant visualizations with a large amount of feature using Mapbox and Postgis. Even though the queries took some time to execute on the testing machine, the performance can theoretically be increased by installing better hardware. By caching the results as tiles they can be transferred and loaded fast on the client on subsequent runs.

RQ3: *What kind of patterns can be revealed and confirmed by a human research utilizing the application?*

Visualizations made by the application reveal known and unknown patterns in the datasets, and more can likely be discovered by more extensive usage and investigation. The patterns are naturally similar to the previous, as they are based on the same data, but a slightly different detail is provided. The visualizations reveal patterns that are very likely to be erroneous, but only can be confirmed through further experiments. Consequently, the application can be used as a tool when developing methods for cleaning the datasets.

RQ4: *Can visualization methods different from the ones used in previous work give new insights?*

Different methods of visualizations can provide new insights. Earlier work have used heat maps which are very good at giving the overall picture, but by using point clouds and trajectories one can get an even more fine-grained and detailed picture where it may be easier to detect low-level issues.

8.2 Future Work

Investigate movements recorded between floors where a trajectory starts in one floor and continues in another. The floors on campus are not necessarily at the same level physically. The first floor of one building may for instance align better with the second floor of another due to inherent geographical phenomena like slopes. In this, it also lies a possibility to find better visualizations of trajectories between floors.

Simpler aggregates for larger time ranges. The results from this work has shown that it is not feasible to provide a snapshot dot map for datasets much larger than up to 10 million points. To provide for such a use case, the points could be clustered into smaller areas and for example colored based on number of points using either squares, circles or hexagons binning approaches. This removes the ability to do fine-grained pattern discovery, but could paint a broader picture of for example seasonal differences or differences between weekdays and weekends. An example query utilizing clustering with PostGIS is given in Appendix B.4.

The origin-destination matrix could be extended from only taking buildings into account, to accept multiple user-defined "objects of interest" such as specific rooms, cafés or other similar points of interest.

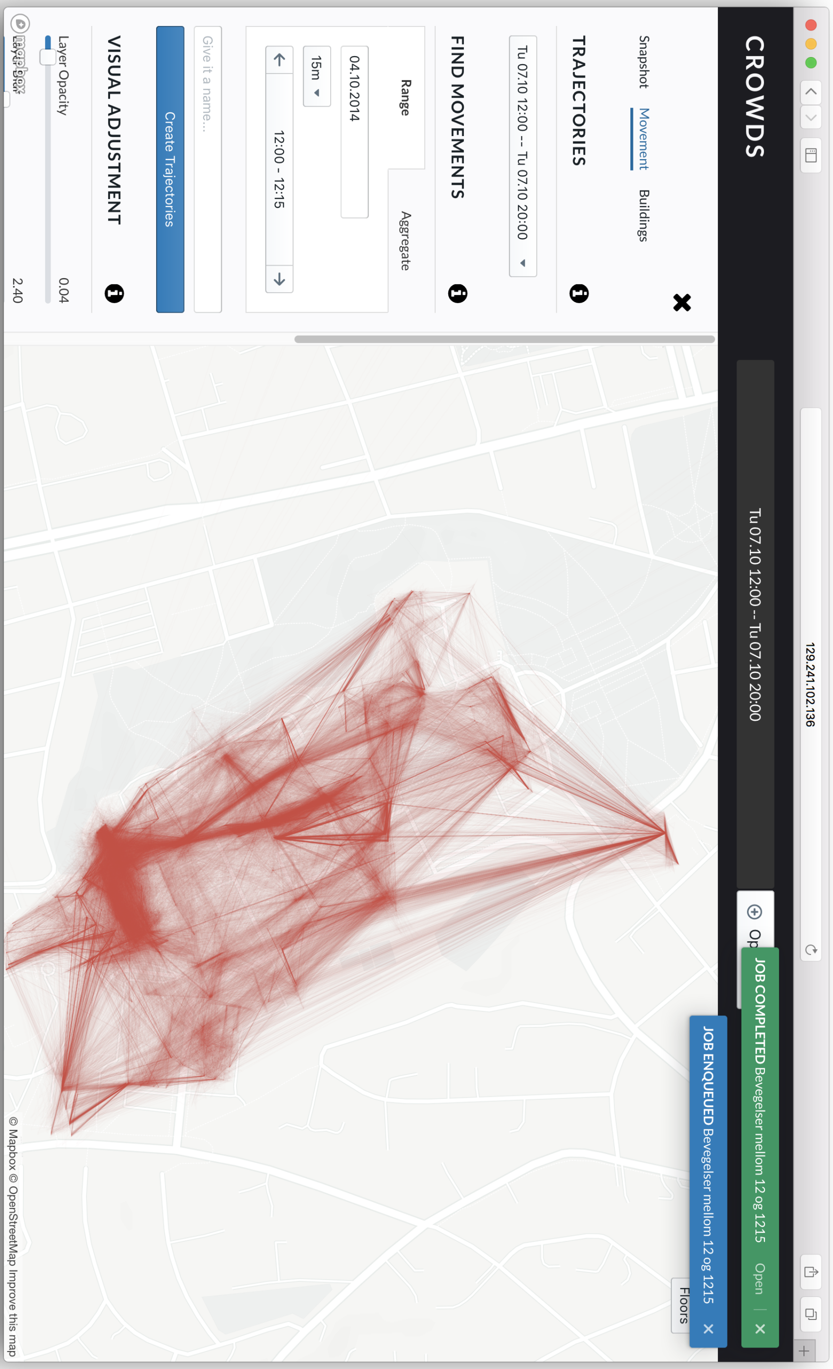
In a meeting with potential interested parties it came up that it would be useful to tailor this application for specific buildings with specialized features, as building managers were often responsible for a single building, or that their main line of work revolved around a particular building.

Compare patterns resulting for the old tracking equipment with results from newer equipment to spot differences and see if there are improvements. This project has presented some patterns from the older equipment and possible reasons, but there should be conducted experiments to compare and thereby increase the trustworthiness of the data.

Finally, it could probably be interesting to tailor the application to real-time use cases where you not only use the historical data, but can visualize data captured in nearly real-time to allow to watch movement patterns around campus in real-time. This would entail building a pipeline for processing new data on the fly and make an adapter for displaying it in the application.

Appendix A

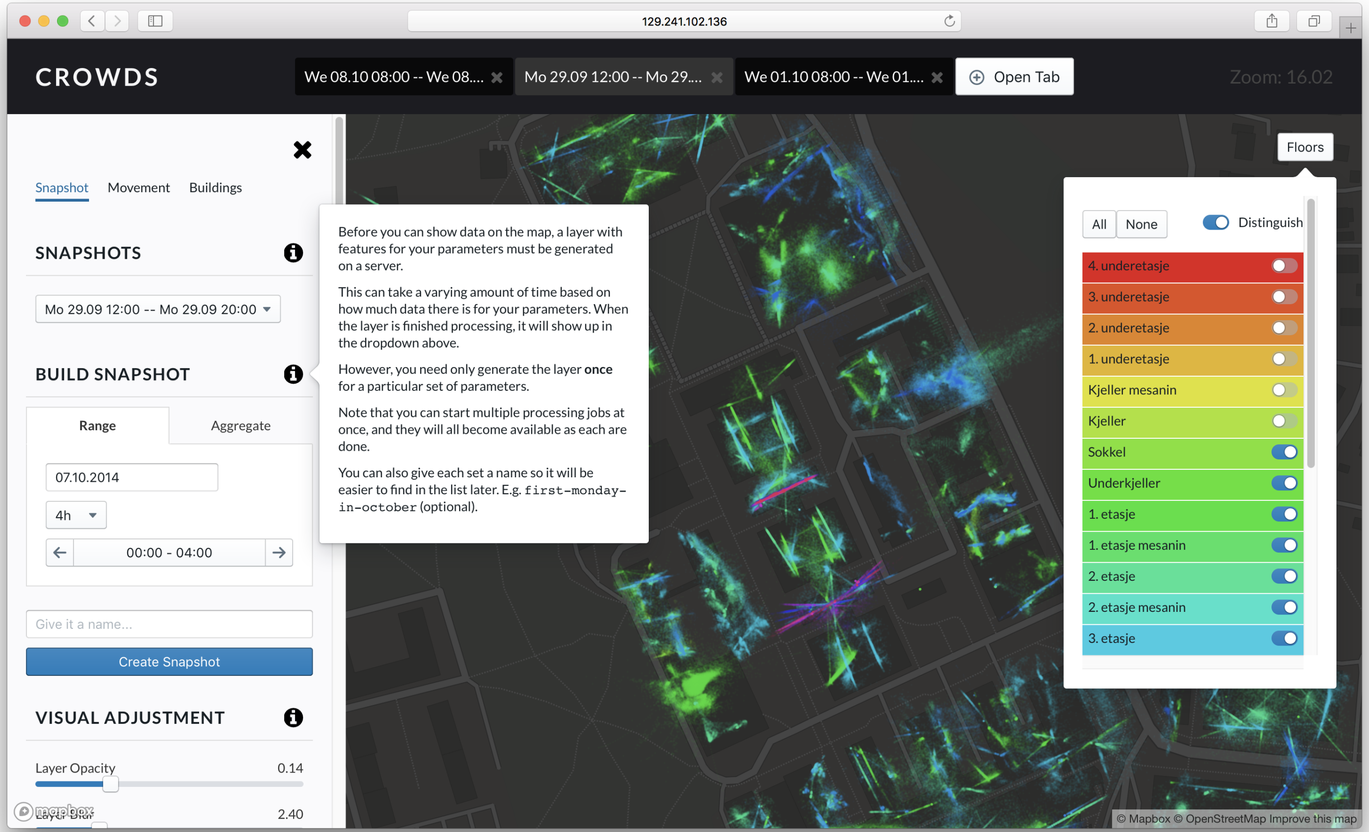
Screenshots of the Application



JOB COMPLETED Bevegelser mellom 12 og 1215 Open

JOB ENQUEUED Bevegelser mellom 12 og 1215

Figure A.1: Trajectory mode with light background map tiles



Before you can show data on the map, a layer with features for your parameters must be generated on a server.

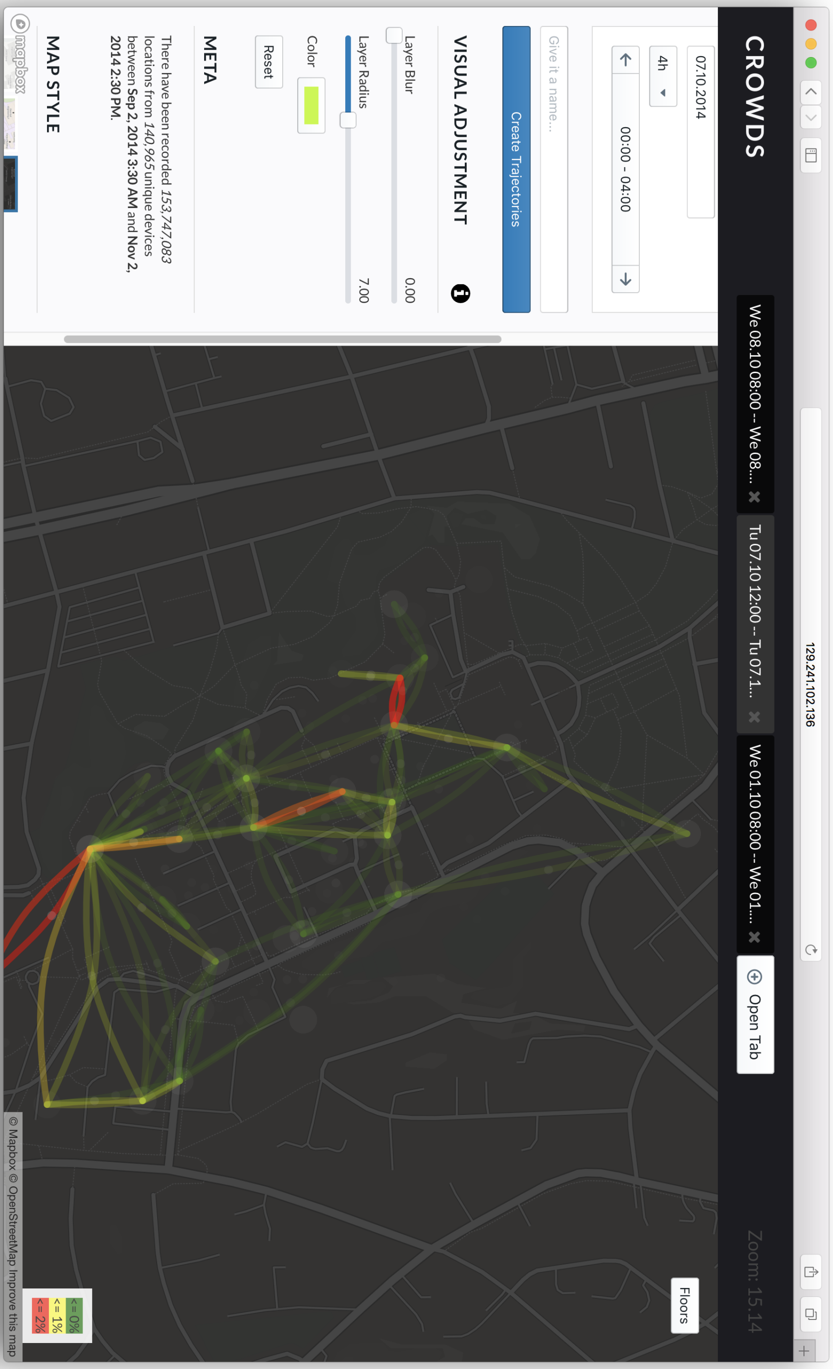
This can take a varying amount of time based on how much data there is for your parameters. When the layer is finished processing, it will show up in the dropdown above.

However, you need only generate the layer **once** for a particular set of parameters.

Note that you can start multiple processing jobs at once, and they will all become available as each are done.

You can also give each set a name so it will be easier to find in the list later. E.g. `first-monday-in-october` (optional).

Figure A.2: Snapshot mode with the floor picker and help text open



129.241.102.136
CROWDS
We 08.10 08:00 -- We 08....
Tu 07.10 12:00 -- Tu 07.1...
We 01.10 08:00 -- We 01....
Open Tab
Zoom: 15.14
07.10.2014
4h
00:00 - 04:00
Give it a name...
Create Trajectories
Floors
<= 0%
<= 1%
<= 2%
© Mapbox © OpenStreetMap Improve this map

VISUAL ADJUSTMENT

Layer Blur 0.00

Layer Radius 7.00

Color

Reset

META

There have been recorded *153,747,083* locations from *140,965* unique devices between **Sep 2, 2014 3:30 AM** and **Nov 2, 2014 2:30 PM.**

MAP STYLE

Figure A.3: Buildings mode showing inter-building movements

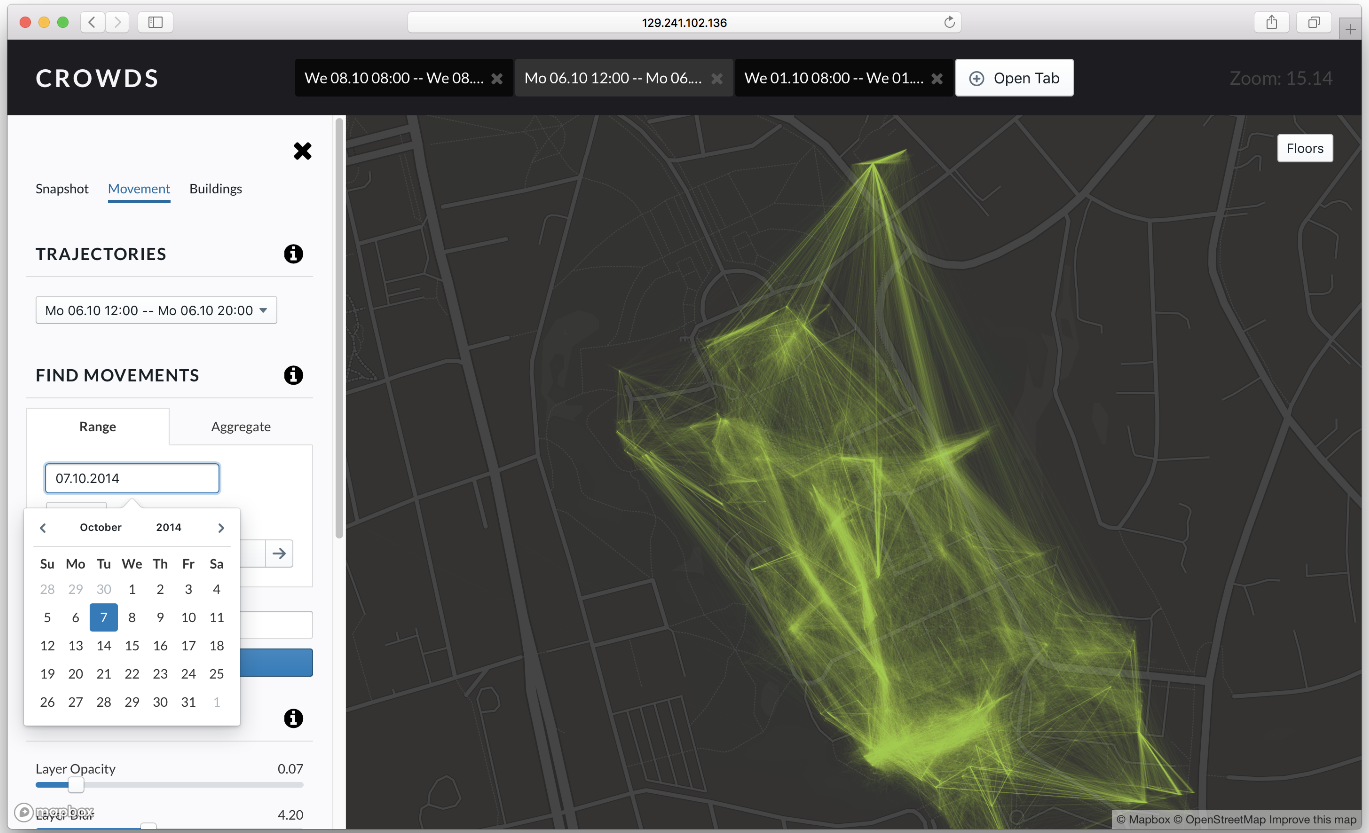
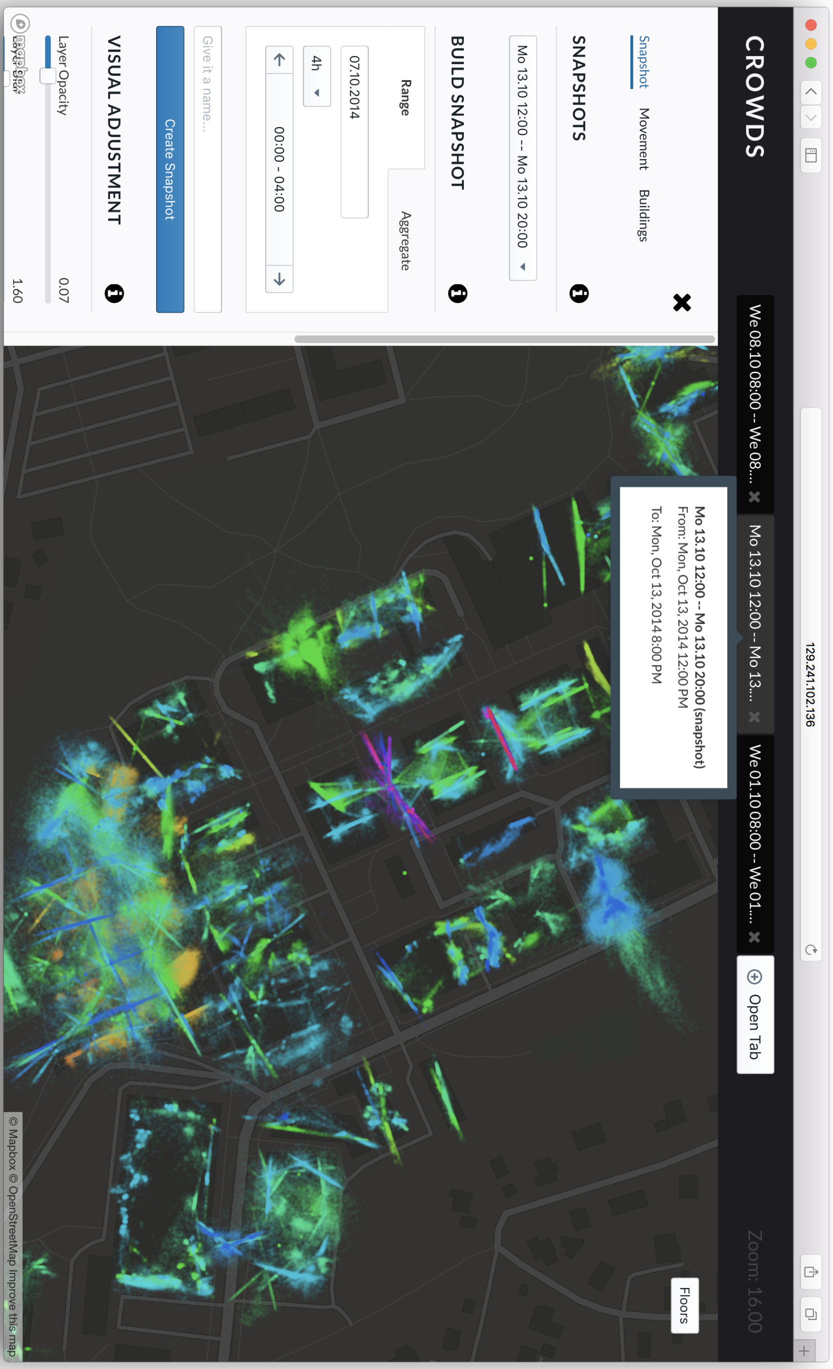


Figure A.4: Movement mode with the default dark background and date picker open



Mo 13.10 12:00 -- Mo 13.10 20:00 (snapshot)
From: Mon, Oct 13, 2014 12:00 PM
To: Mon, Oct 13, 2014 8:00 PM

Figure A.5: Snapshot mode with distinguished floors

Appendix B

Digital Attachments

B.1 Source Code and Figures

The source code for the application and high resolution screenshots are attached as a Zip-file: `attachments.zip`.

B.2 How To Install the System

The following instructions show how to install the system with its dependencies on Ubuntu.

```
# Install Node.js
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg \
| sudo apt-key add -

echo "deb https://dl.yarnpkg.com/debian/ stable main" \
| sudo tee /etc/apt/sources.list.d/yarn.list

sudo apt-get update
sudo apt-get install -y build-essential nodejs yarn

# Install Redis and Postgres
sudo apt-get install -y python-software-properties
sudo add-apt-repository -y ppa:chris-lea/redis
sudo apt-get update
sudo apt-get install -y redis-server
sudo apt-get install -y postgresql postgresql-contrib
sudo -u postgres -i
createuser crowds

# Install Tippecanoe (https://github.com/mapbox/tippecanoe)
sudo apt-get install build-essential libsqlite3-dev zlib1g-dev
git clone git@github.com:mapbox/tippecanoe.git
cd tippecanoe
make
make install

# Download and install dependencies
cd path/to/crowds
yarn

# Create a config file named .env
cat <<EOF > .env
export PGUSER=crowds
export PGDATABASE=posdata_with_acc
export PGPASSWORD=""
export PGHOST=localhost
export PGPORT=5432
export REACT_APP_API_URL=http://localhost:3000
export REACT_APP_MAPBOX_TOKEN="mapbox token from mapbox.com"

EOF

export NODE_ENV=production
source .env

# Build the JS and CSS assets
yarn run build

# Run the system (server + workers)
# Could also use http://supervisord.org to run as a daemon.
NODE_ENV=production PORT=3000 yarn start
```

B.3 Database Schema

```
--- Must enable the PostGIS extension
CREATE EXTENSION postgis;

--- Table structure for our recordings
CREATE TABLE locations(
  id bigserial primary key,
  device_id varchar(100),
  location geometry(POINT),
  latitude float,
  longitude float,
  accuracy float,
  campus varchar(100),
  building varchar(100),
  floor varchar(50),
  hour_of_day smallint,
  day_of_week smallint,
  created_at bigint,
  salt_timestamp bigint
);

--- Add indices for faster queries
CREATE INDEX idx_date ON locations (created_at);
CREATE INDEX idx_floor ON locations (floor);
CREATE INDEX idx_hour_day ON locations(hour_of_day, day_of_week);
CREATE INDEX idx_device ON locations (device_id);
CREATE INDEX idx_building ON locations (building);
CREATE INDEX idx_device_day ON locations(device_id, salt_timestamp);
CREATE INDEX idx_day_hour ON locations(day_of_week, hour_of_day);

--- Clean up some data
UPDATE locations SET floor = '12. etasje' WHERE floor = '12. etasjen'
UPDATE locations SET floor = '1. etasje' WHERE floor = '1. Etasje'

-- This should be run after data is imported to create the PostGIS
-- point geometry from the (lat,lng) tuple.
UPDATE locations SET location = ST_MakePoint(longitude, latitude)
WHERE location IS NULL
```

B.4 Example Queries

```
-- Simple clustering
WITH less_data AS (
  SELECT * FROM locations LIMIT 10000
)
SELECT
  COUNT( location ) AS count,
  ST_AsGeoJSON(
    ST_Expand(
      ST_Centroid(ST_Collect( location )
    ), 0.001)
  ) AS center
FROM less_data
GROUP BY ST_SnapToGrid(location, 0.001)

-- Weekends at 12:00
SELECT *
FROM locations
WHERE day_of_week IN (6,7) AND hour_of_day = 12;
```

Appendix C

List of Technologies

This chapter enumerates all the important frameworks and technologies used in this project along with a short description.

Node.js JavaScript runtime for servers.

PostgreSQL Relational Database System.

Mapbox GL JS JavaScript library using WebGL to render interactive maps.

Webpack Module Bundler for JavaScript and CSS.

React Declarative UI Library for JavaScript applications.

Kue A job queue based on Redis for Node.js.

Redis An in-memory data structure store.

Tippecanoe Build vector tiles from small and large sets of GeoJSON features.

Bibliography

[1] How web maps work. https://www.mapbox.com/help/how-web-maps-work/, Last accessed: 29.05.2017.

[2] React reconciliation. https://facebook.github.io/react/docs/reconciliation.html, Last Accessed: 18.04.2017.

[3] Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, and Christian Tominski. *Visualization of Time-Oriented Data*. Human-Computer Interaction Series. Springer, 2011.

[4] S.H. Andresen, J. Krogstie, and T. Jelle. Lab and research activities at wireless trondheim.

[5] G. Andrienko and N. Andrienko. Spatio-temporal aggregation for visual analysis of movements. In *2008 IEEE Symposium on Visual Analytics Science and Technology*, pages 51–58, Oct 2008.

[6] G. Andrienko, N. Andrienko, P. Bak, D. Keim, S. Kisilevich, and S. Wrobel. A conceptual framework and taxonomy of techniques for analyzing movement. *J. Vis. Lang. Comput.*, 22(3):213–232, June 2011.

[7] Gennady Andrienko, Natalia Andrienko, and Georg Fuchs. Understanding movement data quality. *Journal of Location Based Services*, 10(1):31–46, 2016.

[8] Gennady Andrienko, Natalia Andrienko, Christophe Hurter, Salvatore Rinzivillo, and Stefan Wrobel. Scalable analysis of movement data for extracting and exploring significant places. *IEEE Transactions on Visualization and Computer Graphics*, 19(7):1078–1094, July 2013.

[9] Natalia V. Andrienko and Gennady L. Andrienko. Spatial generalization and aggregation of massive movement data. *IEEE Trans. Vis. Comput. Graph.*, 17(2):205–219, 2011.

[10] Kristoffer Gebuhr Aulie. Human mobility patterns from indoor positioning systems. Master's thesis, NTNU, 2015.

[11] T.A. DeFanti B.H. McCormick and M.D. Brown. Visualization in scientific computing. 21(6), 1987.

[12] Gergely Biczók, Santiago Diez Martinez, Thomas Jelle, and John Krogstie. Navigating mazemap: indoor human mobility, spatio-logical ties and future potential. *CoRR*, abs/1401.5297, 2014.

[13] H. Butler, M. Daly, A. Doyle, Sean Gillies, T. Schaub, and T. Schaub. The GeoJSON Format. RFC 7946, August 2016.

[14] W. Chen, F. Guo, and F. Y. Wang. A survey of traffic data visualization. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):2970–2984, Dec 2015.

[15] CubeWerx. Figure of tiled map. http://www.cubewerx.com/technology/wmts/, Last Accessed: 27.02.2017.

[16] Fred D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Q.*, 13(3):319–340, September 1989.

[17] Ramez A. Elmasri and Shankrant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1999.

[18] Jeppe Benterud Eriksen. Visualization of crowds from indoor positioning data. Master's thesis, NTNU, 2015.

[19] Nivan Ferreira, James T. Klosowski, Carlos Eduardo Scheidegger, and Cláudio T. Silva. Vector field k-means: Clustering trajectories by fitting multiple vector fields. *CoRR*, abs/1208.5801, 2012.

[20] I. Fette and A. Melnikov. The websocket protocol. RFC 6455, RFC Editor, December 2011. http://www.rfc-editor.org/rfc/rfc6455.txt.

[21] Eric Fischer. Mapping extremely dense point data with vector tiles. https://www.mapbox.com/blog/vector-density/, Last Accessed: 20.03.2017.

[22] Eric Gundersen. Announcing mapbox gl js. https://www.mapbox.com/blog/mapbox-gl-js/, Last Accessed: 20.01.2017.

[23] Tor Arne Lyngstad Holten. Evaluation of the accuracy of positioning data for representing room usage. Master's thesis, NTNU, 2016.

[24] H. V. Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M. Patel, Raghu Ramakrishnan, and Cyrus Shahabi. Big data and its technical challenges. *Commun. ACM*, 57(7):86–94, July 2014.

[25] Jan Ježek, Karel Jedlička, Tomáš Mildorf, Jáchym Kellar, and Daniel Beran. *Design and Evaluation of WebGL-Based Heat Map Visualization for Big Point Data*, pages 13–26. Springer International Publishing, Cham, 2017.

[26] Sean Kandel, Jeffrey Heer, Catherine Plaisant, Jessie Kennedy, Frank van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, Dominique Brodbeck, and Paolo Buono. Research directions in data wrangling: Visuatizations and transformations for usable and credible data. *Information Visualization*, 10(4):271–288, October 2011.

[27] Tijmen R. Klein, Matthew van der Zwan, and Alexandru Telea. Dynamic multiscale visualization of flight data. In Sebastiano Battiato and José Braz, editors, *VISAPP (1)*, pages 104–114. SciTePress, 2014.

[28] Magnus Alderslyst Kongshem. Scalable database architecture for human indoor mobility systems. Master's thesis, NTNU, 2016.

[29] Thomas Liebig, Gennady Andrienko, and Natalia Andrienko. Methods for analysis of spatio-temporal bluetooth tracking data. *Journal of Urban Technology*, 21(2):27–37, 2014.

[30] Mapbox. Tippecanoe (software). https://github.com/mapbox/tippecanoe/, Last Accessed: 31.05.2017.

[31] Mapbox. Mbtiles specification. https://github.com/mapbox/mbtiles-spec, Last Accessed: 24.04.2017, 2017.

[32] R. Mautz. *Indoor Positioning Technologies.* Geodätisch-geophysikalische Arbeiten in der Schweiz. ETH Zurich, Department of Civil, Environmental and Geomatic Engineering, Institute of Geodesy and Photogrammetry, 2012.

[33] MazeMap. Mazemap js api docs. http://api.mazemap.com/js/v1.2.1/docs/, Last Accessed: 20.01.2017.

[34] Granville Miller and Laurie Williams. Personas: Moving beyond role-based requirements engineering, 2006.

[35] Rovshen Nazarov and John Galletly. Native browser support for 3d rendering and physics using webgl, html5 and javascript. In Christos K. Georgiadis, Petros Kefalas, and Demosthenes Stamatis, editors, *BCI (Local)*, volume 1036 of *CEUR Workshop Proceedings*, page 21. CEUR-WS.org, 2013.

[36] K. Al Nuaimi and H. Kamel. A survey of indoor positioning systems and algorithms. In *2011 International Conference on Innovations in Information Technology*, pages 185–190, April 2011.

[37] Klokan Petr Přidal. Tiles à la google maps: Coordinates, tile bounds and projection. http://www.maptiler.org/google-maps-coordinates-tile-bounds-projection/, Last Accessed: 20.05.2017.

[38] Arthur A. Shaw and N.P. Gopalan. Finding frequent trajectories by clustering and sequential pattern mining. *Journal of Traffic and Transportation Engineering (English Edition)*, 1(6):393 – 403, 2014.

[39] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, VL '96, pages 336–, Washington, DC, USA, 1996. IEEE Computer Society.

[40] Ben Shneiderman. Extreme visualization: Squeezing a billion records into a million pixels. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 3–12, New York, NY, USA, 2008. ACM.

[41] Hideaki Takeda, Paul Veerkamp, Tetsuo Tomiyama, and Hiroyuki Yoshikawa. Modeling design processes. *AI Mag.*, 11(4):37–48, October 1990.

[42] TMurgent Technologies. White paper: Perceived performance tuning a system for what really matters. 2003.

[43] T. Theoharis, G. Papaioannou, N. Platis, and N. M. Patrikalakis. *Graphics and Visualization: Principles & Algorithms*. A. K. Peters, Ltd., Natick, MA, USA, 2007.

[44] V. Vaishnavi and W. Kuechler. Design research in information systems. http://desrist.org/design-research-in-information-systems/, 2004.

[45] Tatiana von Landesberger, Felix Brodkorb, Philipp Roskosch, Natalia V. Andrienko, Gennady L. Andrienko, and Andreas Kerren. Mobilitygraphs: Visual analysis of mass mobility dynamics via spatio-temporal graphs and clustering. *IEEE Trans. Vis. Comput. Graph.*, 22(1):11–20, 2016.

[46] Vanessa Wang, Frank Salim, and Peter Moskovits. *The Definitive Guide to HTML5 WebSocket*. Apress, Berkely, CA, USA, 1st edition, 2013.

[47] Shi Yin. The state-of-the-art of geographic visualization in urban planning. 2015.