Université Libre de Bruxelles

MEMO-F-524 –Master thesis

Spatiotemporal Analysis of Brussels Mobility Data using MobilityDB

Authors:

Wassim BELGADA

000459836 M-INFO

June 8th

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The contemporary urban landscape is marked by a relentless pursuit of efficient vehicular mobility, essential for the sustenance of economic activities and the seamless functioning of cities. This pursuit, however, faces a formidable challenge in the domain of vehicle detection. As cities grow denser, and urbanization expands its footprint, the traditional methods of detecting vehicles, predominantly optimized for identifying moving entities, reveal their limitations. This becomes particularly evident when addressing the complexities associated with detecting stationary vehicles. The failure to accurately discern non-moving vehicles poses a significant risk to the optimization of traffic flow, responsiveness of emergency systems, and overall efficiency in urban mobility.

In urban environments, the challenge of identifying stationary vehicles is exacerbated by various factors, including diverse parking scenarios, temporary halts, and nuanced traffic patterns. Traditional detection technologies, often reliant on the motion-based paradigm, struggle to adapt to the intricacies posed by stationary vehicles. This limitation not only compromises the accuracy of traffic management systems but also hampers the potential for comprehensive data-driven decision-making in urban planning.

The need to address this deficiency is underscored by the potential ramifications for urban sustainability and safety. Inaccurate or delayed detection of stationary vehicles can lead to traffic congestion, impede emergency response times, and hinder the overall responsiveness of transportation systems. Therefore, there is an urgent demand for research that not only identifies the shortcomings in current detection methodologies but also proposes innovative solutions to enhance the precision and adaptability of vehicle detection systems in mobilitychallenged environments.

1.1 Goals of the Master Thesis

The primary objective of this study is to analyze data provided by Brussels Mobility (BM) to extract valuable insights for enhanced traffic control in Brussels. As the principal regulatory body for traffic management in the region, BM is inherently vested in attaining a high level of real-time knowledge regarding traffic dynamics to facilitate prompt decision-making.

To achieve this overarching goal, two specific objectives have been delineated :

- **Detection of stationary vehicles**
- **Vehicle counting**

These objectives collectively serve the overarching aim of optimizing traffic flow. The first objective is particularly pertinent to corporate entities with a fleet of vehicles, as their larger size can have a disproportionate impact on traffic. The second objective, vehicle counting, provides a quantitative basis for evaluating traffic fluidity. By discerning patterns in vehicle movement, the system can identify areas of congestion and facilitate timely interventions to alleviate traffic issues.

In summary, this research endeavors to employ data analysis techniques to enhance the efficiency of traffic management in Brussels. The specific focus on stationary vehicle detection and vehicle counting aligns with the broader goal of achieving a more responsive and adaptive traffic control system, ultimately contributing to the optimization of urban mobility.

1.1.1 Detection of Stationary Vehicles

The focus here is on identifying vehicles that are stationary, as opposed to those in motion. This aspect is particularly significant for larger vehicles affiliated with specific entities, such as STIB vehicles. Given their typically substantial size, these vehicles can impede traffic flow significantly. The ability to detect stationary vehicles is crucial for proactive traffic management, allowing for swift responses to potential congestion points.

However, pinpointing definitive stops within movement trajectories poses a challenge, primarily due to tracking inaccuracies that result in movement speed rarely registering as true zero. This is particularly evident in GPS tracks, where continuous movement around an object's stop location is common.

The determination of suitable stop definitions is inherently context-dependent. Different applications necessitate varied stop criteria based on specific analytical goals. For instance, an application focused on analyzing trip purposes may prioritize stops of a certain duration, such as those lasting more than 5 minutes. Analysts might seek to infer the purpose of a stop from both its location and duration. Conversely, shorter stops, such as momentary delays at traffic lights, may be deemed irrelevant for the particular objectives of this application.

In essence, the challenge lies in reconciling tracking limitations with the application's specific needs, necessitating a nuanced approach to defining and identifying stops within movement trajectories.

1.1.2 Vehicle Counting

This objective involves quantifying the number of vehicles traversing a specific area. Accurate counting provides critical data for assessing the flow of traffic. An abundance of vehicles passing through a given location suggests smooth traffic conditions, while a lower count may indicate congestion or an issue in proximity to that location. Utilizing this information aids in identifying potential problem areas, contributing to more effective traffic management strategies.

1.2 Contribution of the Master Thesis

Several contributions were made thanks to this work :

- **MobilityDB database for BM**
- **Alarm system for STIB vehicles**
- **Visualization tool for historical data**

1.2.1 MobilityDB Database for Brussels Mobility

In the purpose of this work we needed to create a space of storage for all the data that will be analyzed in real-time and to store the historical data in order to allow BM analyze the traffic in past and understand how a situation has appeared.

The database used in our case is MobilityDB and more details about it will be given in Section 2.2

1.2.2 Alarm System for STIB Vehicles

To allow BM to quickly react when a vehicle is not moving an blocking the circulation, we have implemented an alarm system that will periodically go through the trips of all the vehicle and detect whether or not a vehicle is not moving anymore. But as explained in section 1.1.1 the problem is more complicated than it looks like.

The alarm will be verified and if it is accepted it will be loaded on the Mapboard which is a complicated tool developed by BM to allow them to handle all the traffic on the main roads of Brussels.

1.2.3 Visualization Tool for Historical Data

This contribution consist into the implementation of a tool that would allow the user to visualize historical data from the STIB vehicles by choosing a day and an hour and the replay how were the vehicles moving.

This implementation will not be a definitive version since there are some issue that we cannot handle and that are described in section 3.3.6

Chapter 2

State of the Art

2.1 M3

M3 is a mobility software dedicated and developed by Macq company. The company has been involved in mobility for 25 years. This software allows for various functionalities:

- Identify (ANPR), recognize (brand, model, color), and classify all types of vehicles (heavy/-light traffic, trucks, buses, cars, motorcycles, bicycles, etc.).
- Counting all types of vehicles.
- Creation of Origin-Destination Matrices for all vehicles circulating in defined areas. For example, a set of intersections.
- Measurement of the travel time of vehicles.

The data collected by the cameras is based on vehicle license plates. They are anonymized in the M3 server before being sent to the various clients who will use them for their needs.

The server behind the software currently provides data with a delay of 5 minutes, which can vary depending on the client's expectations and goals, but this requires having the necessary infrastructure to be able to receive this data. We are talking about 2 million data entries per day, which is likely to increase significantly with the addition of new cameras in the city of Brussels in the coming months.

Nowadays Macq provide 3 types of data sources to BM that allows to have a counting of vehicles which are :

- **G3 camera** : Can differentiate a car from a truck.
- **G5 camera** : Can also recognize the bran of the vehicle.
- **OBU Viapass** :
- **Tomtom data** : The opendata provided by tomtom users.

The precision of the cameras makes them a good source of data for analyzing vehicle counts. For camera G3, 97% of the vehicles are correctly recognized and 95% of them are well identified by OCR reading. For G5 camera 99% of the vehicles are correctly recognized and 99.5% of them are well identified by OCR reading.

2.2 Moving Object Databases (MOD)

A moving object database is a database management system that allows for the storage and query of objects that change over time. First we had spatial and temporal databases, and now we have the fusion of the two which is MOD (moving object database). In order for the extension to MODs to be done, a temporal or spatial variable must be added to already existing variables. Several cases already exist in which the use of MOD is the best choice. Among them we find delivery applications, in which the customer likes to know where his delivery man has arrived at any time and thus can have his position in real-time on the application, or the big export companies which would like to follow in real-time the position of their ship to act as fast as possible in case of problem. The use cases are very varied, ranging, as we have already seen, from delivery to the tracking of planes or ships or even to follow the evolution of something much more important, like a cyclone.

MobilityDB is a moving object database that is implemented as an extension to PostgreSQL and PostGIS that provides the necessary database support for storing and querying geospatial trajectory data. It implements persistent database types, and query operations for managing geospatial trajectories and their time-varying properties [1].

MobilityDB continues to show its efficiency, as for example by showing that it is efficient on distributed systems [**bakli2019distributed**] or that it allows to analyse trajectories of moving objects [**rovinelli2021multiple**].

[1] [**MobilityDBTODS2020**]

2.3 Mapbox Vector Tile

The Mapbox Vector Tile (MVT) specification, using the Google Protobuf format (PBF), represents a significant advancement in the handling and presentation of geospatial data. This format is notably efficient and compact, which is particularly beneficial for web and mobile applications where performance and responsiveness are key considerations.

The Mapbox Vector Tile Specification details how to encode geographic information into vector tiles. These tiles do not inherently store geographic coordinates like latitude and longitude; instead, they encode points, lines, and polygons as x,y pairs relative to the top left of the grid in a right-down manner. This encoding transforms geographic coordinates into vector tile grid coordinates, significantly optimizing data size and processing speed.

Attributes within the vector tiles are encoded using a series of tags that reference keys and values, providing a streamlined representation of the original key:value pairs from the geometry. This method efficiently handles large geometries by reducing redundancy in attributes with the same keys and similar values.

The winding order, which refers to the direction in which a ring is drawn (clockwise or counter-clockwise), is a crucial aspect of the specification. It is vital for renderers to distinguish between unique geometries and holes within multipolygons. According to the specification, exterior rings must be oriented clockwise, and interior rings must be oriented counter-clockwise.

A diverse range of applications and tools have adopted the Mapbox Vector Tile Specification. These include parsers, generators, client-based web tools, browser-based applications, and servers. Notable implementations of MVT can be seen in various sectors. For instance, The New York Times has used Mapbox Tiling Service (MTS) for creating detailed election maps. TripAdvisor's Crowdfree has developed a social distancing index, and companies like Stubhub and Polaris Ride Command have integrated MTS for enhancing their user experiences. These implementations highlight the versatility and robustness of the MVT in handling complex geospatial data and rendering it in a user-friendly manner.

2.4 PG_TILESERV

pg_tileserv, as of 2024, is a specialized PostGIS-only tile server that has been designed with a focus on simplicity and efficiency. It's written in Go, which contributes to its lightweight and efficient nature.

There are several key features and aspects in its architecture thas make it a candidate for tile serving, among them :

- **Exclusive Use of PostGIS** : pg_tileserv is exclusively dedicated to serving vector tiles from PostGIS data sources. This singular focus on PostGIS allows for several advantages:
  - Automatic Configuration: The server can automatically discover and publish tile sources from all tables it has read access to in a PostgreSQL/PostGIS database.
  - Full SQL Flexibility: It allows the use of any SQL command to generate tile outputs, enabling advanced data processing, feature filtering, and record aggregation.
  - Database Security Model: pg_tileserv supports standard database access control, including advanced techniques like row-level security, to dynamically filter access based on the login role.
- Architecture: pg_tileserv is part of the ”PostGIS for the Web” (PostGIS FTW) ecosystem, a collection of Go-based spatial microservices. This ecosystem revolves around a PostgreSQL/PostGIS database cluster, allowing for a stateless microservices architecture that reduces middleware software complexity. In this ecosystem:
  - pg_tileserv provides MVT (Mapbox Vector Tile) tiles for interactive clients and rendering.
  - pg_featureserv offers GeoJSON feature services for reading and writing vector and attribute data from tables.
- **Functionality and Usage**: pg_tileserv operates by taking in HTTP tile requests and executing SQL, with an API and design inspired by the Martin tile server. It's built to facilitate the installation and use in spatial applications and supports various mapping frameworks like OpenLayers, Leaflet, and Mapbox GL JS.
- **Licensing and Community Support**: pg_tileserv is available under the Apache 2.0 License, indicating its open-source nature. The development and maintenance of pg_tileserv are supported by Crunchy Data, with the team welcoming contributions, questions, and issue reports through their GitHub page.

This makes pg_tileserv an effective tool for those looking to serve vector tiles directly from PostGIS without the need for additional data sources or complex configurations. Its integration with the broader PostGIS ecosystem also ensures that it can be part of a powerful and flexible web-based geospatial platform.

2.5 Transit Information Standards

2.5.1 NeTEx

NeTEx (Network Timetable Exchange) is a European technical standard for sharing public transport data, including network topology, scheduled timetables, and fare information. It is based on Transmodel and outlines the format and protocol for exchanging passenger information like stops, routes, timetables, and fares. While it expanded to include shared mobility as of 2020, its primary focus remains on static data describing services and infrastructure, leaving real-time information to SIRI.

Developed and maintained by the Committee for Standardisation (CEN), NeTEx is overseen by Technical Committee 278 (TC278), Working Group 3 (WG3), Sub Group 9 (SG9). Its purpose is to harmonize and ensure cross-border interoperability within the European Union, replacing national standards that were originally designed for operational purposes and unsuitable for transnational interoperability.

NeTEx comprises:

- A CEN Specification document.
- A data model using the UML modeling language.
- An XML schema providing a formal electronic description for data processing software. Data in NeTEx format is encoded as XML documents, with standard XML validator tools ensuring conformance. The schema can also generate bindings for various programming languages.

NeTEx serves as the EU standard for exchanging static scheduled data, encompassing public transport, long-distance coach, and maritime transport, including ferries. It's freely available under the Apache 3.0 license and is mandated for use by EU national access points (NAPs) as specified in the ITS Directive.

Unlike GTFS, NeTEx does not cover passenger information, eliminating privacy concerns. NeTEx has a broader scope than GTFS, serving both operational management and customer-facing systems, accommodating complex use cases. GTFS focuses on providing essential information in a straightforward manner, while NeTEx is designed to handle a wide range of scenarios.

Governance of NeTEx involves consensus-building within a working group comprising experts nominated by CEN national members, representing national stakeholders. The final draft undergoes a voting process based on weighted votes proportional to the member state's population, similar to Transmodel's governance structure.

2.5.2 SIRI

SIRI (Service Interface for Real-time Information) is the European standard for sharing real-time information about public and shared transport services and vehicles. It facilitates the exchange of structured real-time data concerning schedules, vehicles, connections, vehicle availability, parking, and operational messages.

Developed and maintained by the Committee for Standardisation (CEN), SIRI falls under Technical Committee 278 (TC278), Working Group 3 (WG3), Sub Group 7 (SG7). Its primary purpose is to complement NeTEx by providing real-time data to NeTEx's scheduled information. Both SIRI and NeTEx share a common conceptual model based on Transmodel, enabling data exchange between operators, within operator systems, and with passenger information systems.

Technically, SIRI comprises various services, each with a specific function:

- Production Timetable Service: Exchanges planned schedules for a specific day, including updates, often shared with NeTEx and used by Automatic Vehicle Management Systems (AVMS).
- Estimated Timetable Service: Shares estimated real-time schedules and updates for AVMS

systems.
- Stop Timetable Service: Provides schedule information for arrivals and departures at a stop.
- Stop Monitoring Service: Offers arrivals and departures information from a stop-centric perspective.
- Vehicle Monitoring Service: Gives details about vehicle movements and progress against schedules.
- Connection Timetable Service: Provides interchange schedule information.
- Connection Monitoring Service: Supports guaranteed connection services.
- General Message Service: Exchanges general text messages related to stops, lines, etc.
- Situation Exchange Service: Covers the exchange of incident information and planned events affecting public transport, optimized for journey planners.
- Facility Monitoring Service: Shares facility status information, particularly its impact on persons with reduced mobility (PRMs).

Due to its comprehensiveness, SIRI requires the definition of an implementation profile that meets specific needs. While it complements NeTEx, SIRI is designed as a standalone solution, making real-time data processing somewhat heavier and less efficient compared to formats like GTFS Real-time, which are more suitable for web application development.

SIRI is the EU standard for exchanging dynamic public transport data, mandated for use by EU national access points (NAPs) as of 2019, as specified in the ITS Directive. For passenger-oriented information, GTFS Real-time is commonly used. GDPR compliance is expected given its official EU-standard status, although specific requirements may vary with the introduction of new modes like car and bike sharing. No notable privacy or GDPR issues have been identified in research.

Governance of SIRI follows a consensus-building process within a working group comprising experts nominated by CEN national members. Experts represent national stakeholders, and the final draft undergoes a voting process based on weighted votes proportional to member state populations, similar to Transmodel's governance structure.

2.5.3 GTFS

GTFS (General Transit Feed Specification) is a data format that enables public transit agencies to share their transit information with various software applications. It consists of a schedule component containing information about schedules, fares, and geographic data, and a real-time component with arrival predictions, vehicle positions, and service advisories. MobilityData, a French-Canadian non-profit organization, currently hosts and maintains GTFS.

Originally developed by TriMet in 2005 to integrate public transit schedules into Google Maps, GTFS was designed to simplify communication of service information for passenger journey planning. Over time, it has been adopted by multiple cities and applications, becoming a widely accepted standard.

GTFS comprises two parts:

GTFS Schedule: This component consists of text files in CSV format containing relatively stable transit data such as stops, routes, trips, and schedule information. Updates are typically made every few months. GTFS Realtime: GTFS Realtime includes three binary files with real-time vehicle positions, arrival information, and service alerts, updated approximately every minute. Implementing GTFS Realtime can be more technically challenging, particularly for smaller transport agencies. GTFS is designed to be user-friendly for both humans and machines. While it does not provide exhaustive operational details like NeTEx, it offers sufficient information for trip planning and timetables for public transport riders. Thousands of organizations use GTFS, making it a de facto standard for consumer-oriented public transport information. Even organizations using NeTEx may use GTFS for consumer applications.

GTFS is compatible with NeTEx for schedule information and SIRI for real-time information. It can be generated from NeTEx or SIRI data, but not the other way around. GTFS is not affected by privacy-related legislation like GDPR since it does not handle sensitive data.

The governance of GTFS is open, with input and changes proposed by the transit community, developers, and stakeholders. Variants of GTFS, such as GTFS-ride and GTFS-flex, offer additional functionalities like standardized ride data collection and modeling of Demand-Responsive Transportation (DRT) services, expanding its utility beyond fixed-route public transportation.

2.5.4 DATEX II

The origins of DATEX II can be traced back to the early 1990s when European countries recognized the need for a common framework to facilitate the exchange of traffic-related data. Prior to DATEX II, the absence of standardized data formats hindered cross-border cooperation and the development of harmonized traffic management and information systems. DATEX II was officially initiated in 2003 under the auspices of the European Commission. A consortium of experts and stakeholders from across Europe collaborated to develop a standardized language for traffic data exchange. Over the years, DATEX II has evolved into a mature and widely adopted standard, encompassing diverse traffic data domains.

A large range of traffic data categories can be covered with DATEX II, including :

- Traffic Management: Information related to traffic monitoring, control, and regulation.
- Traffic Events: Real-time reporting of incidents, roadworks, and other events affecting traffic flow.
- Traveler Information: Provision of real-time information to travelers, including route guidance and traffic conditions.
- Parking Information: Data on parking facilities, availability, and pricing.
- Roadside Equipment: Information from roadside sensors, cameras, and other monitoring devices.
- Environmental Data: Air quality, weather conditions, and other environmental factors influencing traffic.

2.6 Kafka

Apache Kafka, a leader in real-time data streaming and event processing, has become an indispensable component in modern data architectures. It offers scalable, fault-tolerant solutions to manage high volumes of data streams, making it highly relevant across various industries.

Kafka's architecture is based on a publish-subscribe model, where data producers send information to topics, and consumers retrieve it from these topics. This decoupled system allows for real-time data distribution and scalability, addressing the needs of large-scale data handling efficiently.

Beyond being a mere message broker, Kafka is part of a broader ecosystem, including Kafka Connect for data integration, Kafka Streams for real-time data processing, and Kafka MirrorMaker for cross-data-center replication. Kafka Streams and KSQL (Kafka SQL) serve as potent tools in this landscape. Kafka Streams enables the development of applications for in-cluster data processing, while KSQL provides a more approachable means of stream processing with SQL-like queries.

In microservices architectures, Kafka is pivotal, facilitating inter-service communication and managing event-driven actions within distributed systems. As Kafka's adoption increases, so does the emphasis on security. Modern implementations incorporate robust security features like authentication, authorization, and encryption. These features make Kafka suitable for compliance-sensitive sectors, including finance and healthcare.

Kafka's role extends into machine learning and AI, where it supports real-time data ingestion and distribution, crucial for model training and inference. The Kafka community, including significant contributions from Confluent, the company co-founded by Kafka's original creators, continuously enhances Kafka's capabilities. The Confluent Platform extends Kafka's functionality with additional tools and features.

Despite its widespread success, Kafka faces challenges like the need for improved management and monitoring tools, adapting to evolving data formats, and scalability. The future trajectory of Kafka likely involves advancements in user-friendliness, ecosystem component enhancements, and deeper integration with emerging technologies such as Kubernetes and cloud-native architectures.

In summary, Apache Kafka stands out as a high-throughput, scalable solution for real-time data processing and event streaming. Its widespread adoption is a testament to its reliability, flexibility, and robust security features. As Kafka continues to evolve, it is expected to address current challenges and maintain its position as a cornerstone technology in the field of data streaming and processing.

Chapter 3

Data Sources and Manipulation

3.1 Server

Thanks to BM's infrastructure, it was possible for me to set up a machine within their server, allowing us to collect the data that will be used later for analysis. It also helps on-site operators manage real-time traffic using real-time data and historical data if a study on a past situation were to take place.

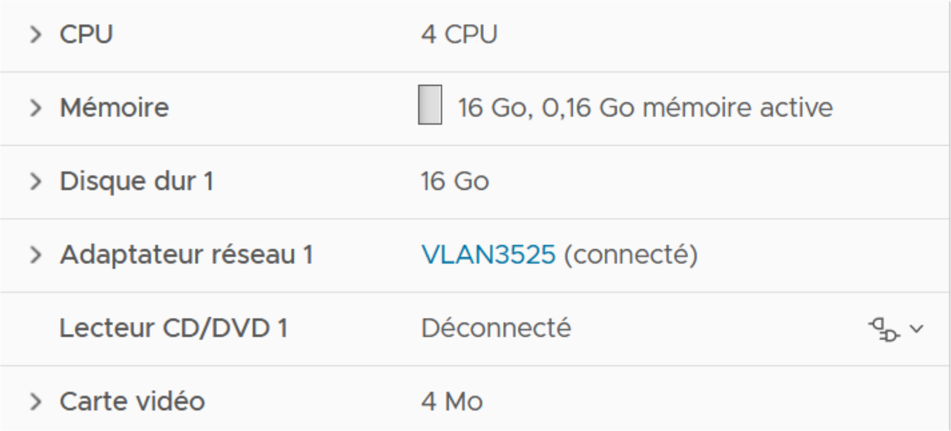
The machine used for data collection has the characteristics shown in figure 3.1.

To ensure the security and maintainability of this machine, we install an AV and the VMWare Tools. Since the machine belong to BM which is a governmental organ the security aspect is very important, and the VMWare tools are used to ensure set of services and modules that enable several features in VMware products for better management of, and seamless user interactions with, guests operating systems.

3.2 Data Collection

In order to collect the data, we wrote the code for a server that makes requests to various APIs we want to access, which have interesting information.

The different APIs are as follows:



> CPU	4 CPU
> Mémoire	16 Go, 0,16 Go mémoire active
> Disque dur 1	16 Go
> Adaptateur réseau 1	VLAN3525 (connecté)
Lecteur CD/DVD 1	Déconnecté
> Carte vidéo	4 Mo

Figure 3.1: Server in the PaaS

- STIB Open Data Portal : For all real-time data and static GTFS data.
- TLC API : For data concerning traffic lights.
- M3 API : For counting data provided by ANPR cameras.

Additional details to be added later, which I haven't thought of yet.

3.3 STIB

3.3.1 Open Data Portal (ODP)

There are different sources of data provided to us through the STIB open data portal. As we can see in the figure 3.2, it is possible to link a large portion of the data to which we have access in order to cover a maximum number of use cases that may interest us.

We will review the various tables to determine what is at our disposal and what we will use.

3.3.1.1 Vehicle Position RT

This part of the API provides us with significant information for the study of mobility, namely the last known position of all circulating STIB vehicles at any given moment.

The data is provided in JSON format and follows a schema as follows:

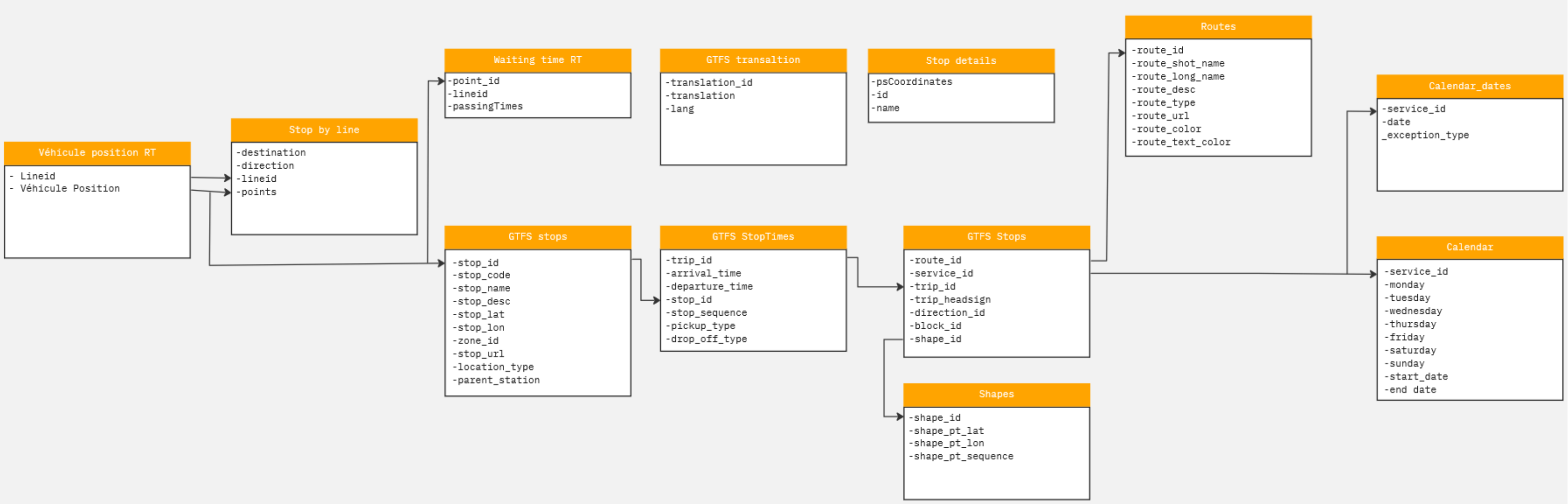
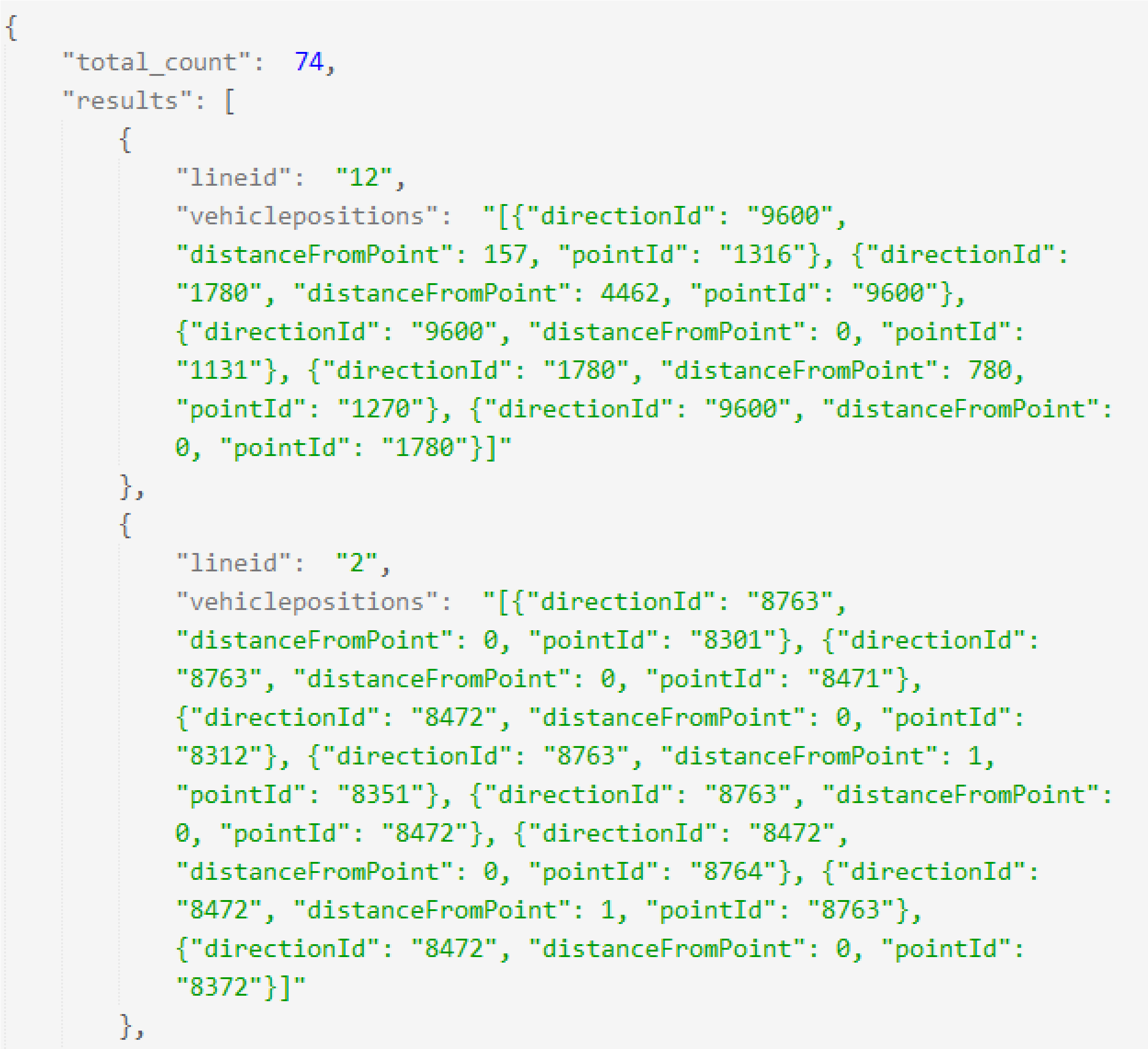


Figure 3.2: Data available on the open portal of the STIB



```
{
    "total_count":  74,
    "results": [
        {
            "lineid":  "12",
            "vehiclepositions":  "[{"directionId": "9600",
            "distanceFromPoint": 157, "pointId": "1316"}, {"directionId":
            "1780", "distanceFromPoint": 4462, "pointId": "9600"},
            {"directionId": "9600", "distanceFromPoint": 0, "pointId":
            "1131"}, {"directionId": "1780", "distanceFromPoint": 780,
            "pointId": "1270"}, {"directionId": "9600", "distanceFromPoint":
            0, "pointId": "1780"}]"
        },
        {
            "lineid":  "2",
            "vehiclepositions":  "[{"directionId": "8763",
            "distanceFromPoint": 0, "pointId": "8301"}, {"directionId":
            "8763", "distanceFromPoint": 0, "pointId": "8471"},
            {"directionId": "8472", "distanceFromPoint": 0, "pointId":
            "8312"}, {"directionId": "8763", "distanceFromPoint": 1,
            "pointId": "8351"}, {"directionId": "8763", "distanceFromPoint":
            0, "pointId": "8472"}, {"directionId": "8472",
            "distanceFromPoint": 0, "pointId": "8764"}, {"directionId":
            "8472", "distanceFromPoint": 1, "pointId": "8763"},
            {"directionId": "8472", "distanceFromPoint": 0, "pointId":
            "8372"}]"
        },
```

Figure 3.3: Vehicles position in RT

The JSON file provides us with various pieces of information:

- **total_count** : The total number of lines with at least one vehicle in circulation.
- **results** : The set of results for each STIB line.

For each STIB line, we have various pieces of information:

- **lineid** : the number of the line
- **vehiclepositions** : The set of positions for each vehicle on the line.

As we can see in the image, the data is separated by line. For each line, we have three pieces of information for each vehicle that is moving on the line at the moment the query was made.

The information is as follows:

- **directionId** : The ID of the terminus of the line towards which the vehicle is heading.
- **distanceFromPoint** : This is the distance in meters that the bus has traveled since the last stop it passed.
- **pointID** : This is the ID of the last stop that the vehicle passed.

3.3.1.2 Stop by line

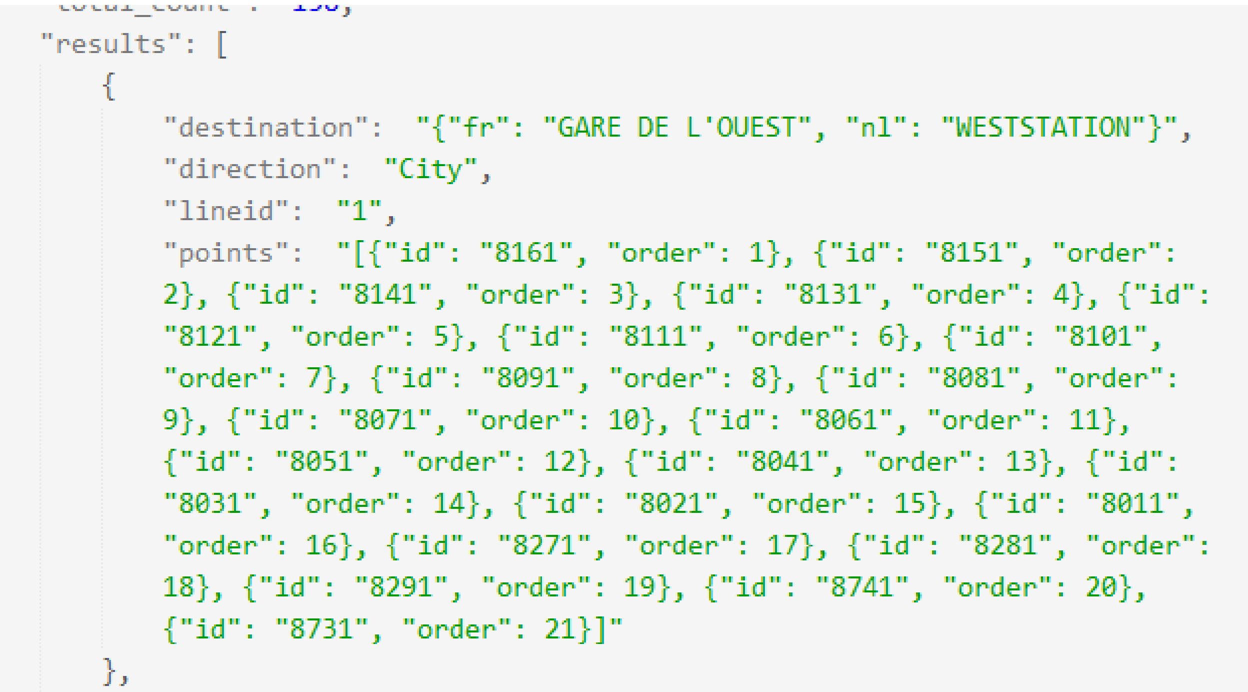
This table allows us to know, for each line, the stops in the order in which they are taken by the various vehicles of the STIB.

As can be seen in Figure 3.4, we have different data fields for each line:

- **destination** : the terminus of the line.
- **direction** : where the line ends.
- **points** : A list of data pairs with the stop number in order and its ID.

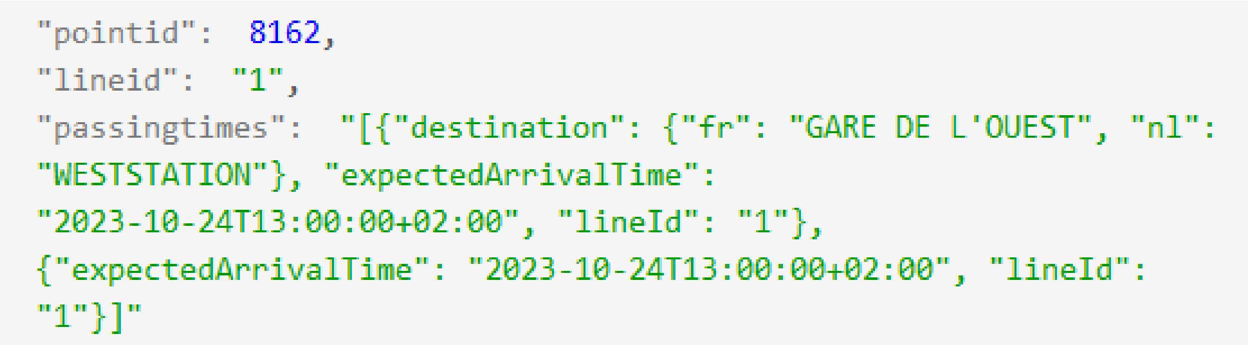
3.3.1.3 Waiting times RT

This dataset allows us to know how soon the next two vehicles of the line will arrive for each stop on the STIB network.



```
 "total_count": 198,
 "results": [
     {
         "destination":  "{"fr": "GARE DE L'OUEST", "nl": "WESTSTATION"}",
         "direction":  "City",
         "lineid":  "1",
         "points":  "[{"id": "8161", "order": 1}, {"id": "8151", "order":
         2}, {"id": "8141", "order": 3}, {"id": "8131", "order": 4}, {"id":
         "8121", "order": 5}, {"id": "8111", "order": 6}, {"id": "8101",
         "order": 7}, {"id": "8091", "order": 8}, {"id": "8081", "order":
         9}, {"id": "8071", "order": 10}, {"id": "8061", "order": 11},
         {"id": "8051", "order": 12}, {"id": "8041", "order": 13}, {"id":
         "8031", "order": 14}, {"id": "8021", "order": 15}, {"id": "8011",
         "order": 16}, {"id": "8271", "order": 17}, {"id": "8281", "order":
         18}, {"id": "8291", "order": 19}, {"id": "8741", "order": 20},
         {"id": "8731", "order": 21}]"
     },
```

Figure 3.4: Stops by line request result



```
"pointid":  8162,
"lineid":  "1",
"passingtimes":  "[{"destination": {"fr": "GARE DE L'OUEST", "nl":
"WESTSTATION"}, "expectedArrivalTime":
"2023-10-24T13:00:00+02:00", "lineId": "1"},
{"expectedArrivalTime": "2023-10-24T13:00:00+02:00", "lineId":
"1"}]"
```

Figure 3.5: Waiting times RT request result

As we can see in Figure 3.5, the different data fields obtained are as follows:

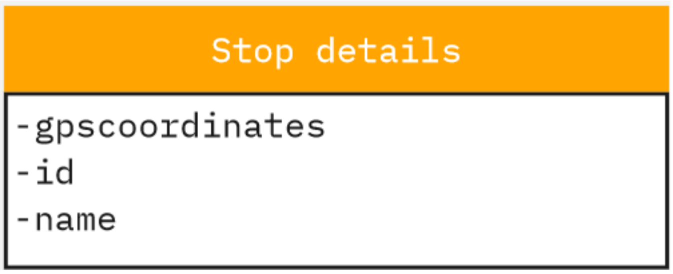
- **destination** : The stop at which the vehicles will arrive.
- **lineid** : The id of the line
- **passingTimes** : The estimated arrival time for the next two vehicles at the stop specified in the "destination" field.

3.3.1.4 GTFS Dataset

GTFS, which stands for General Transit Feed Specification, is a standardized format used to describe public transportation systems, making it easy for developers, transit agencies, and third-party applications to access and display information about public transit services. GTFS primarily consists of a set of plain text files containing structured data that provides a clear and comprehensive representation of a public transportation system.

Here are the different data that follow the standard of GTFS that STIB provide through the open data portal.

- **agency.txt** : This file contains details about the transit agency, including its name, URL, and contact information. It establishes the agency's identity within the dataset.
- **stops.txt** : Lists all the stops within the transportation network, including their unique IDs, names, coordinates (latitude and longitude), and stop codes.
- **routes.txt** : Describes the various routes operated by the agency. It includes route names, IDs, and other attributes such as route type (bus, subway, tram, etc.).
- **trips** : Defines the specific trips or journeys that vehicles make on each route. It associates a trip with a route, service, and other information.
- **stop_times.txt** : Contains detailed scheduling information for each trip, listing the times when vehicles arrive and depart from each stop. It includes arrival and departure times, stop IDs, and sequence information.
- **calendar.txt** : Provides a calendar of service, specifying the days of the week when services are active and date ranges for service exceptions like holidays.
- **calendar_dates.txt** : Lists dates that have exceptions to the regular service schedule, indicating dates when a specific service is added or excluded.
- **shapes.txt** : Contains the spatial data representing the geographic paths of routes, often used for mapping and visualization purposes.



Stop details

-gpscoordinates
-id
-name

Figure 3.6: Stop details table

3.3.1.5 Stop details

This data set is almost the same than the data provides in the "**stop.txt**" file of the GTFS dataset. The difference is that there is less details, the only that availlable for each stop are:

- **gpscoordinates** : the coordinate that we store in the WKB format after creating a PostGIS point with the longitude and latitude provided in the data.
- **stopid** : The id of the stop as in the stops data of the GTFS dataset
- **name** : The french translation of the stop's name

The reason of the usage of both the stop details table provided by the STIB ans the stops table from the GTFS data set will be given and explained later.

3.3.2 Storage

For STIB, real-time data is updated every 20 seconds, and considering that we take the values available at each update, we would need storage space equivalent to that listed in the table 3.1.

We can already observe with the table above a difference in size between the two data sources. This is easily explained by the fact that on one hand, we have information concerning all vehicles in circulation, while on the other hand, it involves all stops in the STIB network, resulting in a significant difference in the size of the two datasets.

Data	20 seconds	1 minute	1 hour	1 day	1 month	1 year
Positions RT	37-60 Ko	111-180 Ko	6.7-10.8 Mo	160.8-259.2 Mo	4 Go-7.8 Go	48-93.6 Go
Waiting Times RT	1.15 Mo	3.46 Mo	208 Mo	4.75 Go	142.7 Go	1.67 To

Table 3.1: Growth of dataset size gradually over time

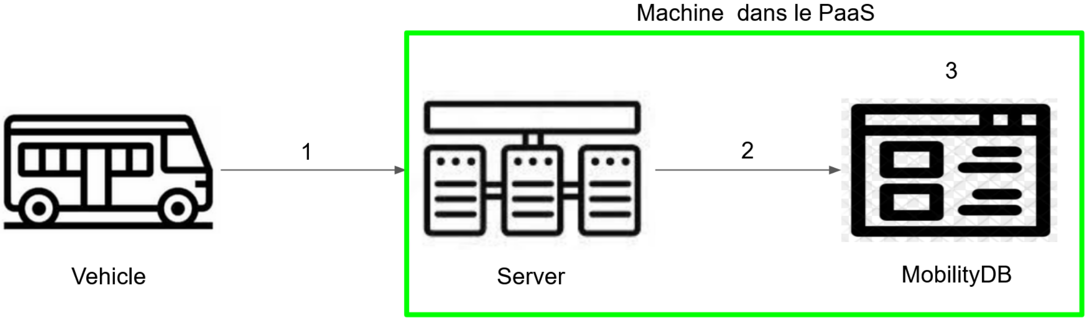


Figure 3.7: Process of STIB data

3.3.3 Process

The process of data collection and processing is visible in the image 3.7, and the different steps will be explained in the following sections.

There are three steps :

- **1** : This simply involves the sending of data by the vehicles and their retrieval via the STIB ODP.
- **2** : Data collection via an API request that then executes queries on the database.
- **3** : Update of the mobilityDB trips

As indicated in Figure 3.7, both the server and the database are installed on a machine on the PaaS network provided by BM. The characteristics of the latter are detailed in Figure 3.1.

3.3.4 Real-Time Position and Reverse Engineering

One of the data sources that interests us the most is the one that allows us to know the real-time position of STIB vehicles. These are updated once every 20 seconds, in accordance with the information provided by the provider.

The data is provided as shown in Figure 3.3.

So, a question arises: What does the **distanceFromPoint** data really represent? Two possibilities are available to us:

- **Rayon de cercle** : This is the as-the-crow-flies distance at which the vehicle is located relative to the last stop it was at.
- **Trajectoire** : This is the number of meters traveled by the vehicle following exactly the trajectory defined in the GTFS dataset shapefiles of the STIB.

In the first case, we could obtain quite interesting information, but it would have lower precision since we cannot locate the vehicle accurately.

In the second case, we would have much more precision as we can implement a system allowing us to calculate the almost exact position of the vehicle in terms of x and y coordinates through interpolation.

The method used to determine which solution was more plausible is as follows.

Firstly, data collection was necessary. Therefore, we ran a data catcher for several hours to obtain a sample of data as varied as possible.

Now we needed to find a section of the route taken by a STIB vehicle where the trajectory is as non-linear as possible for the test to be as reliable as possible. The chosen section is shown in Figure 3.8, which is the route between the stops **Les 3 Tilleuls** and **Miraval**.

By sorting the captured data, we could make a selection to view only the data of the buses on the section of interest. All of this can be done using Excel with some fairly simple formulas. The result obtained is visible in Figure 3.9.

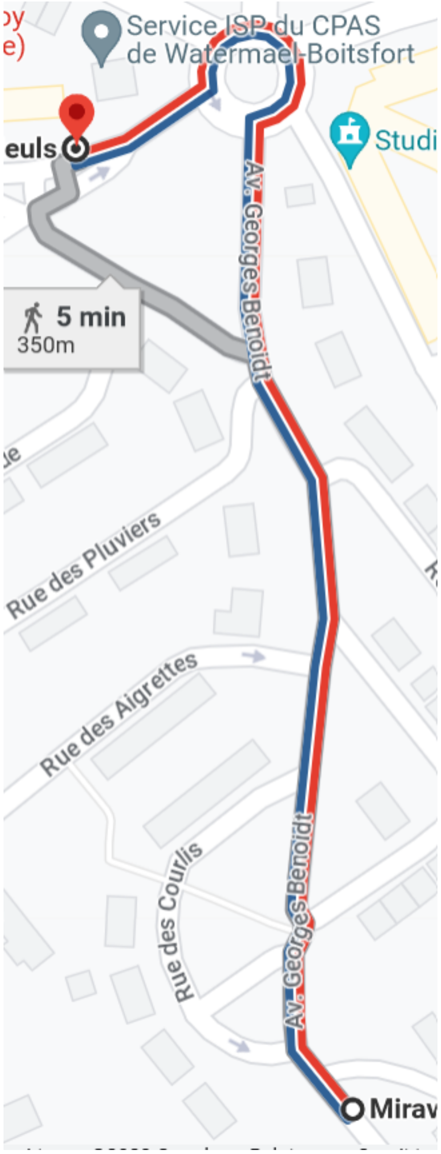
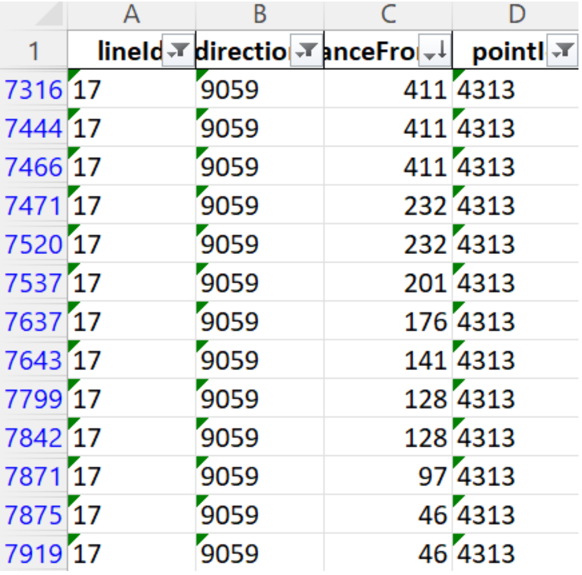


Figure 3.8: Trancon trajet bus 17 et 95



	A	B	C	D
1	lineId	directio	anceFro	pointI
7316	17	9059	411	4313
7444	17	9059	411	4313
7466	17	9059	411	4313
7471	17	9059	232	4313
7520	17	9059	232	4313
7537	17	9059	201	4313
7637	17	9059	176	4313
7643	17	9059	141	4313
7799	17	9059	128	4313
7842	17	9059	128	4313
7871	17	9059	97	4313
7875	17	9059	46	4313
7919	17	9059	46	4313

Figure 3.9: Data recolted

Let's take the largest recorded value and compare it to the distance between the two stops. The distance between the two stops can be read on Figure 3.10. We end up with a difference of two meters between the recorded data and the value that can be found using Google Maps.

Now that we know exactly what the **distanceFromPoint** data represents, namely the number of meters traveled on the predefined trajectory of the vehicle, we can apply the proposal made earlier. Namely, an interpolation that will allow us to obtain the exact coordinates of the vehicle for a more interesting analysis of its behavior using the functionalities available in MobilityDB.

3.3.5 Interpolation and Trip Construction

In this section, we will discuss the method used to construct MobilityDB trips based on the data made available to us on the STIB ODP.

3.3.5.1 Interpolation

The data received via the STIODP from the production of real-time vehicle positions is not sufficient to allow us to construct a MobilityDB trip. Therefore, we need a way to derive real coordinates from the three fields we receive in the result of our API query.

We start with three pieces of information:

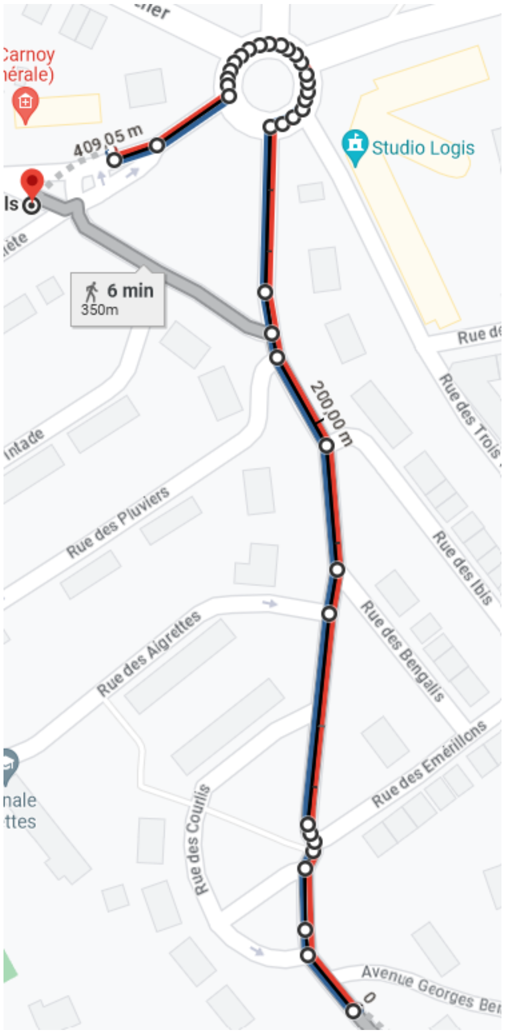


Figure 3.10: Trancon trajet bus 17 et 95

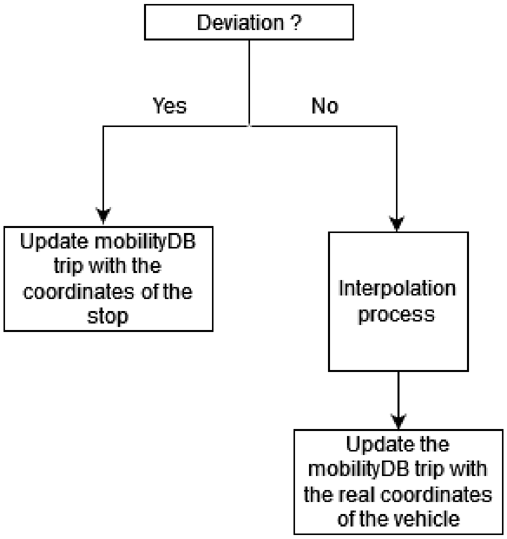

Deviation ?
Yes
No
Update mobilityDB trip with the coordinates of the stop
Interpolation process
Update the mobilityDB trip with the real coordinates of the vehicle

Figure 3.11: Diagram montrant le choix en cas de déviation.

- **directionId** : The id of the terminus
- **distanceFromPoint** : Distance traveled since the last point following the predefined trajectory of the vehicle's route.
- **pointId** : The ID of the stop from which the distance was measured.

As we can see in Figure 3.12, it is challenging to locate the vehicle on the predefined trajectory with just a distance. This is where the interpolation calculation comes in, which will allow us to obtain GPS coordinates from RT data. Before performing the interpolation calculation, it is necessary to first check whether the vehicle is in deviation or not. In the event that it is, we cannot carry out the interpolation due to a lack of information. The process is illustrated in Figure 3.11.

To do this, we need to make certain queries to obtain everything that is necessary.

The first step is to isolate the part of the trajectory between the last visited stop and the end of the line. Therefore we need to find the coordinates of the last visited stop. To do this, we use the query 3.1.

Listing 3.1: Query to find the right id for the terminus and the coordinates for the stop

```

--get the real id terminus
SELECT stopid as stop from stopdetails where stopid LIKE ’%’ || directionID
    || ’%’
```

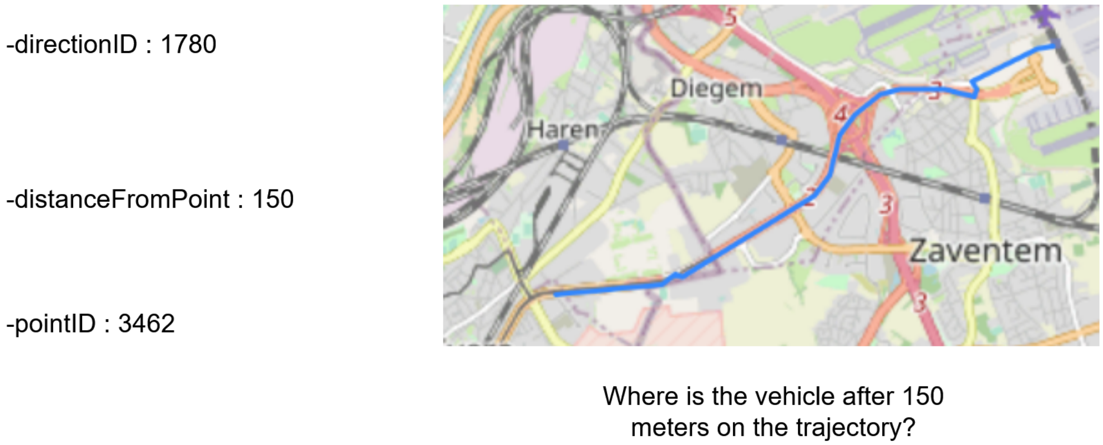


Figure 3.12: Description of the problem when receiving the data

```
--Get coordinate of the stop
SELECT ST_SetSRID(ST_MakePoint(p1.stop_lon,p1.stop_lat),4326) as point1
FROM stops as p1
WHERE p1.stop_id LIKE '%' || pointID || '%' --Replace by pointID
```

Once the terminus and the point are found, we can select the shape associated with the route using the ID of the terminus and the fraction of the route using the point of the last visited stop. This selection of the fraction is done using queries 3.2 and 3.3.

Listing 3.2: Query to find the start of the subtrajectory

```
SELECT
ST_LineLocatePoint(trajectory.trajectory, point1) AS start_fraction
FROM
terminus_shapes AS trajectory
WHERE
trajectory.route_short_name = lineID
  AND trajectory.stopid = directionID
```

Listing 3.3: Query that select the subfraction of the trajectory

```
SELECT
ST_SetSRID(ST_LineSubstring(trajectory.trajectory, strat_fraction,1),4326)
   AS sub_trajectory
FROM
terminus_shapes AS trajectory
WHERE
```

```
trajectory.route_short_name = lineID AND trajectory.stopid = directionID
limit 1
```

With these queries, we can isolate the section of the route that allows us to find the real coordinates of the vehicle through interpolation.

Interpolation is done by executing the query 3.4.

Listing 3.4: Query that calculate the interpolation

```
SELECT ST_LineInterpolatePoint(st.sub_trajectory,
    distanceFromPoint/ST_Length(st.sub_trajectory::geography))
FROM sub_trajectory as st
```

Once all these queries are executed, we have real coordinates that allow us to construct our MobilityDB trips.

It is interesting to see now how trips are updated in the database. Regarding the insertion of a new trip or the update of a trip that has already started, this is done using the function3.5

Listing 3.5: Function to update the table

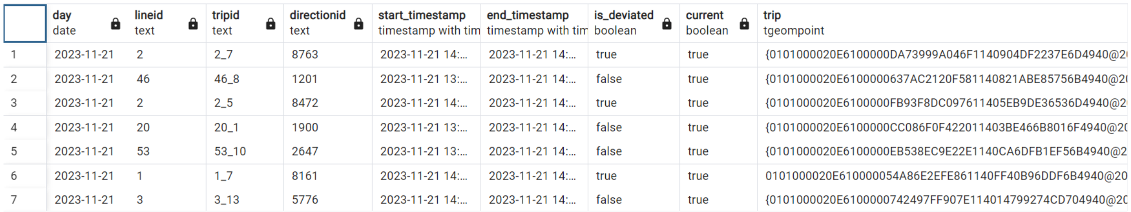
```
CREATE OR REPLACE FUNCTION insert_or_update_stib_trip(
p_day DATE, p_lineid TEXT, p_tripid TEXT, p_directionid TEXT,
p_start_timestamp TIMESTAMPTZ, p_end_timestamp TIMESTAMPTZ,
p_is_deviated BOOLEAN, p_current BOOLEAN, p_trip tgeompoint
)
RETURNS VOID AS $$
BEGIN
-- Check if there is an existing record with the same day, lineid, and tripid
PERFORM 1
FROM stib_trips
WHERE day = p_day AND lineid = p_lineid AND tripid = p_tripid AND
    directionid = p_directionid AND current;

IF FOUND THEN
-- Update the existing record
UPDATE stib_trips
SET
end_timestamp = p_end_timestamp,
is_deviated = p_is_deviated,
trip = update_trip(p_trip, stib_trips.trip)
WHERE
day = p_day AND lineid = p_lineid AND tripid = p_tripid AND directionid = p_directionid AND current;
ELSE
-- Set current to false for the previous record with the same day, lineid, and tripid
UPDATE stib_trips
SET
current = false
WHERE
day = p_day AND lineid = p_lineid AND tripid = p_tripid;

-- Insert a new record
INSERT INTO stib_trips (day, lineid, tripid, directionid, start_timestamp, end_timestamp, is_deviated, current, trip)
VALUES (p_day, p_lineid, p_tripid, p_directionid, p_start_timestamp, p_end_timestamp, p_is_deviated, p_current, p_trip);
END IF;
END;
$$ LANGUAGE plpgsql;

-- Define a function to calculate the updated value for trip column
CREATE OR REPLACE FUNCTION update_trip(new_tgeompoint tgeompoint, existing_tgeompoint tgeompoint)
RETURNS tgeompoint AS $$
DECLARE
updated_tgeompoint tgeompoint;
BEGIN
SELECT appendInstant(existing_tgeompoint, new_tgeompoint) INTO updated_tgeompoint;
RETURN updated_tgeompoint;
END;
$$ LANGUAGE plpgsql;
```

Here, the goal is to add a flag to each trip to indicate whether it is the current journey or not, using the **current** column in the table. To conclude the trip at the right moment, we



	day	lineid	tripid	directionid	start_timestamp	end_timestamp	is_deviated	current	trip
	date	text	text	text	timestamp with tim	timestamp with tim	boolean	boolean	tgeompoint
1	2023-11-21	2	2_7	8763	2023-11-21 14:...	2023-11-21 14:...	true	true	{0101000020E6100000DA73999A046F1140904DF2237E6D4940@2
2	2023-11-21	46	46_8	1201	2023-11-21 13:...	2023-11-21 14:...	false	true	{0101000020E6100000637AC2120F581140821ABE85756B4940@2
3	2023-11-21	2	2_5	8472	2023-11-21 14:...	2023-11-21 14:...	true	true	{0101000020E6100000FB93F8DC097611405EB9DE36536D4940@2
4	2023-11-21	20	20_1	1900	2023-11-21 13:...	2023-11-21 14:...	false	true	{0101000020E6100000CC086F0F422011403BE466B8016F4940@2
5	2023-11-21	53	53_10	2647	2023-11-21 13:...	2023-11-21 14:...	false	true	{0101000020E6100000EB538EC9E22E1140CA6DFB1EF56B4940@2
6	2023-11-21	1	1_7	8161	2023-11-21 14:...	2023-11-21 14:...	true	true	0101000020E610000054A86E2EFE861140FF40B96DDF6B4940@20
7	2023-11-21	3	3_13	5776	2023-11-21 14:...	2023-11-21 14:...	false	true	{0101000020E6100000742497FF907E114014799274CD704940@2

Figure 3.13: Tables containing mobilityDB trips

use the **directionId** field of each vehicle received in the real-time data as the flag.

To achieve this, we construct a map with the trip ID as the key and the terminus towards which the vehicle is heading as the value. In this way, when we receive real-time data and observe a change in the directionId of a vehicle, the flag that designates the current route is reversed, and a new line is added and marked as the one to be updated later.

The trips are stored, as we can see in Figure 3.13.

The reason why the trips have been stored in this way is that it allows for a better analysis in case of issues on the network. Isolating each round trip between the two terminus of a line makes it easy to identify which trip it is by using the vehicle's start and end timestamps.

In the table, we find the following attributes:

- **day** : The day on which the trip was recorded.
- **lineid** : The id of the line.
- **tripid** : This is the vehicle ID, simply the concatenation of the line ID with the vehicle number in the real-time data received.
- **directionid** :The ID of the terminus towards which the vehicle is heading.
- **start_timestamp** : The timestamp from which the trip started to be recorded.
- **end_timestamp** : The timestamp indicating the end of the trip.
- **is_deviated** : A boolean indicating whether there is a deviation on the line during the recording period of the trip.
- **current** : The boolean that serves as a flag to determine if the trip to be updated is the correct one.
- **Trip** : The tgeompoint used for the analysis of vehicle movement data.

3.3.6 Encountered Problems

We encountered various problems during the analysis of STIB data, which means that the obtained result is not entirely real or that we cannot achieve a certain degree of precision in our data.

The first problem encountered, which poses an issue in the construction of MobilityDB trips, is the absence of IDs to distinguish the different vehicles circulating on the STIB network.

The second problem is the absence of certain stops in the STIB network's stop dataset. This is problematic for interpolation calculations because the inability to find a stop prevents us from selecting the correct shape for interpolation.

The last problem is the lack of compatibility between real-time and static data provided in the GTFS dataset. This difference means that some deviations are impossible to reconstruct in MobilityDB trips because the corresponding shapes are not available.

3.4 TLC API

In this section we will discuss about the data that TLC provides to BM in order to visualize the state of traffic lights at any moment anywhere in Brussels.

In the development of analytical tools utilized within this research, a critical data acquisition process is implemented, initiated by an application program interface (API) call to the designated traffic portal. This initial request, as designed by a contributor to the project named Sven, is required to specify the 'CLE'—a concise identifier utilized within Scala programming—and a commencement timestamp. The provision for an optional termination timestamp is also incorporated to delineate the temporal scope of the data retrieval (Sven, 2023) as shown in Figure 3.14 and Table 3.2.

Subsequent to the initiation request, the traffic portal conducts a validation of both the 'CLE' and the temporal parameters. This step is preliminary to the procurement of comprehensive data from the suite of detectors and signal groups that correspond to the control point of interest, spanning the requested time frame. The culmination of this process is the generation of a data file, formatted in either JSON or XML as per the established agreement, which encapsulates the gathered data (Sven, 2023).

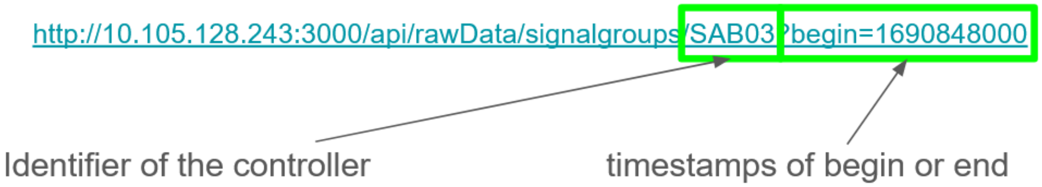

http://10.105.128.243:3000/api/rawData/signalgroups/SAB03?begin=1690848000
Identifier of the controller
timestamps of begin or end

Figure 3.14: TLC request example

Name	Type	Default	Required
begin	Unix Timestamp	Now - 1 hour	False
end	Unix Timestamp	Now	False

Table 3.2: Parameter that can be added to the API request

It is pertinent to note that the data originates from the Scala database (IG), with an imposed latency of five minutes on the most recent records, thereby excluding data within this quintuple-minute timeframe from the retrieval process.

The operational remit of the traffic portal extends to the translation of the API call into the OCIT-C protocol. This translation is accompanied by a series of verifications pertaining to the authenticity of the request and the availability of the data. The final phase involves the portal's conversion of the OCIT-C data response, as received from the IG, back into a user-friendly JSON or XML format [1]. All this process can be visualized in the Figure 3.15.

Once all the process is completed,we receive the response in the format of a JSON file. The metadata of the responsone is give in the Table 3.3 and an image of the JSON response is showed in Figure 3.16.

For the moment any work has been done with this data and every thing will be explained in Section 5.4 from why we cannot work on it for the moment and what could be done with this data.

3.5 M³ API

In this section we will take a look to de that that is provided by the ANPR cameras that we talked about Section 2.1. First of all let's take a look at the process to retrieve the data

[1] This informations are in a word file that Sven from BM shared with us

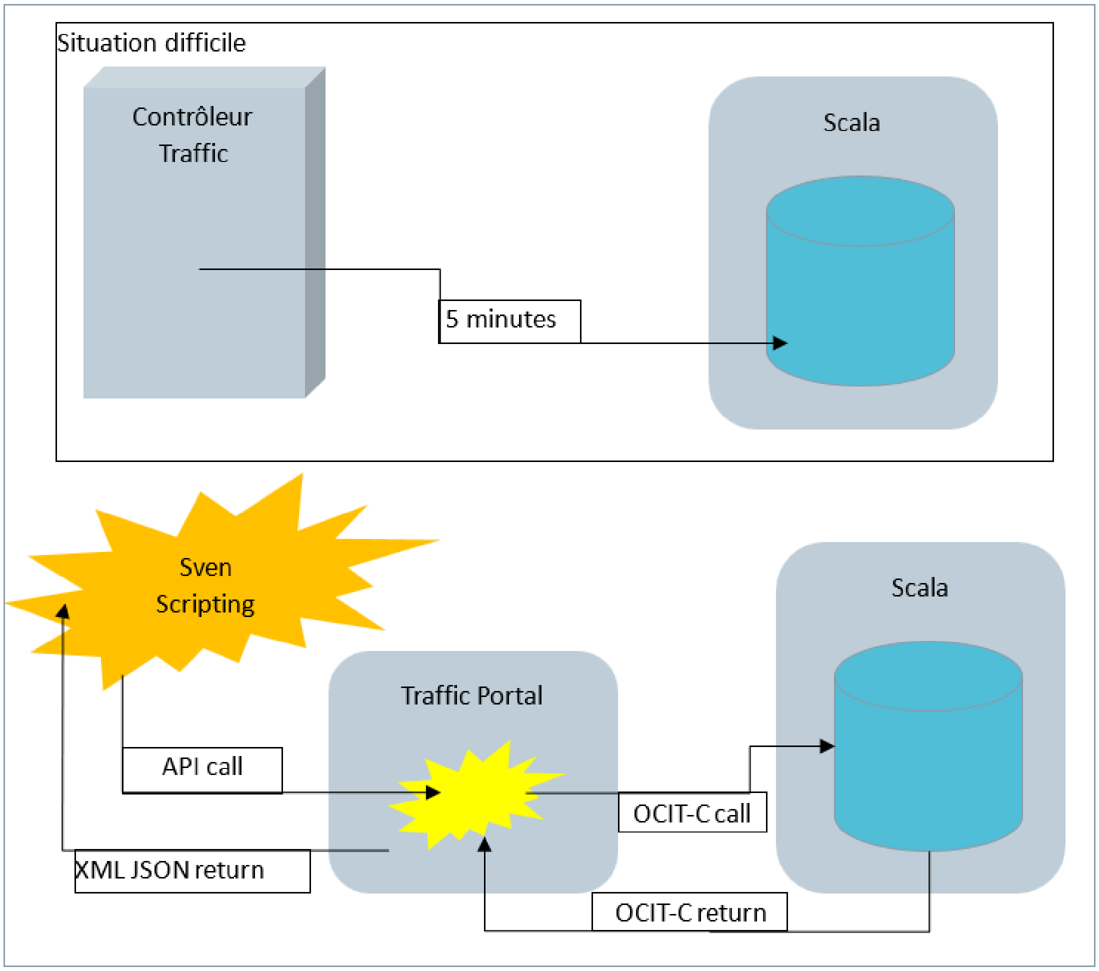
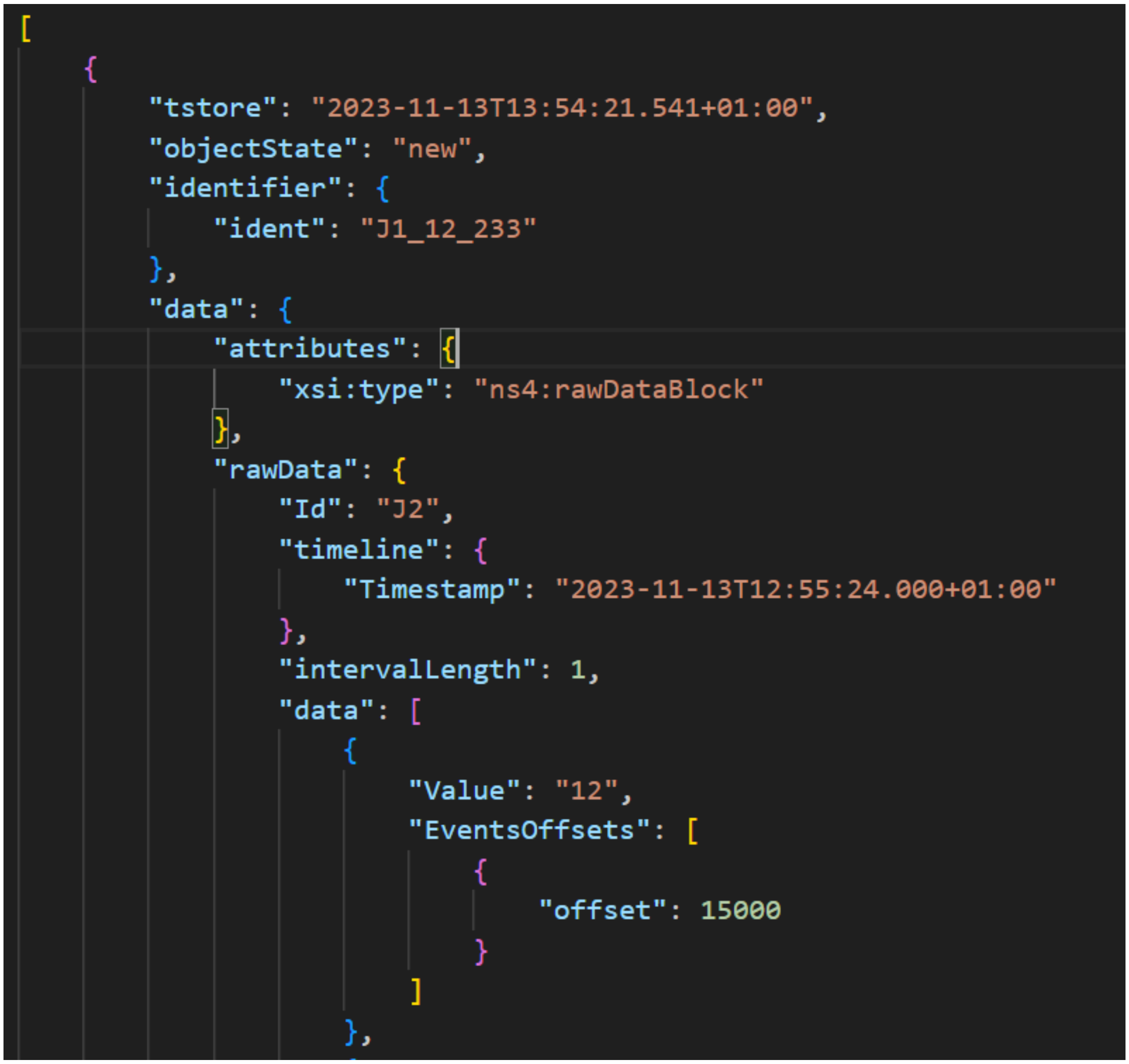


Figure 3.15: Entire process between requesting data and receiving response.



```
[
    {
        "tstore": "2023-11-13T13:54:21.541+01:00",
        "objectState": "new",
        "identifier": {
            "ident": "J1_12_233"
        },
        "data": {
            "attributes": {
                "xsi:type": "ns4:rawDataBlock"
            },
            "rawData": {
                "Id": "J2",
                "timeline": {
                    "Timestamp": "2023-11-13T12:55:24.000+01:00"
                },
                "intervalLength": 1,
                "data": [
                    {
                        "Value": "12",
                        "EventsOffsets": [
                            {
                                "offset": 15000
                            }
                        ]
                    },
```

Figure 3.16: JSON file received as response from TLC API request

Field	**Type**	**Description**	**Example**
tstore	DateTime	Moment the data was stored	2021-04-01T18:43:31.648+02:00
identifier/ident	String	Central intersection No	J1_12_40
rawData ID	String	Name of the signal group	"Id": "J8"
Timestamp	DateTime	Holds the measurement timestamp	2021-04-02T07:38:59.270+02:00
intervalLength	Int	Time unit in milliseconds	1
Data Value	Int	Identifies the state of the detector: • 0: black • 3: red • 12: orange • 15: Red-Orange • 32: green blinking • 48: green	Value: "3" Value: "48"
Data Offset	Int	milliseconds after the timeline timestamp where the detector state changed	"offset": 33230

Table 3.3: Data Structure Description

as it is showed in Figure 3.17, we can see that kafka is used to stream the data and this is explained by all the reasons given in Section 2.6.

There is two kind of data that are provided by M^3, the first categorie is the counting of vehicle around a camera and the second one is the statistics about the average speeds of the vehicles between two cameras. Both file a represented by examples in Figures 3.18 and 3.19

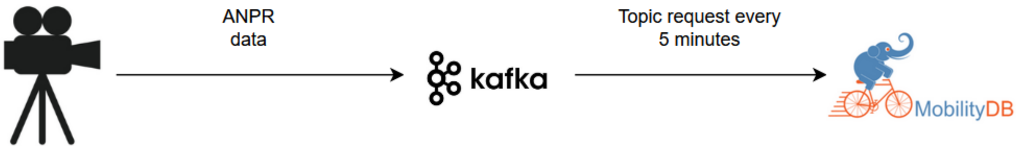
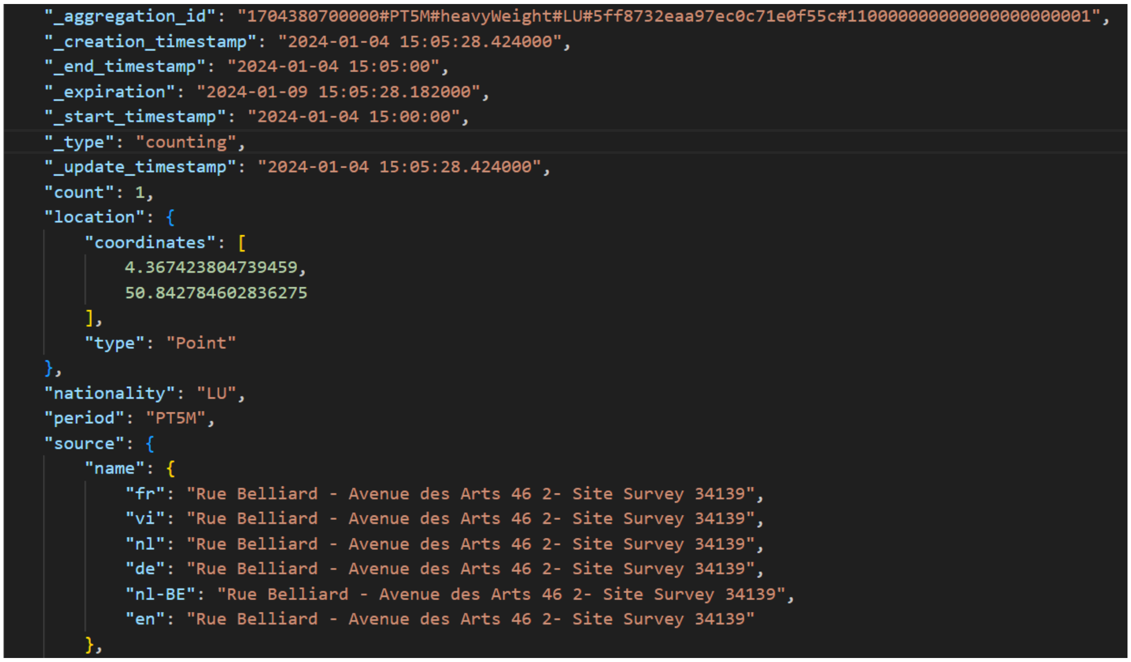
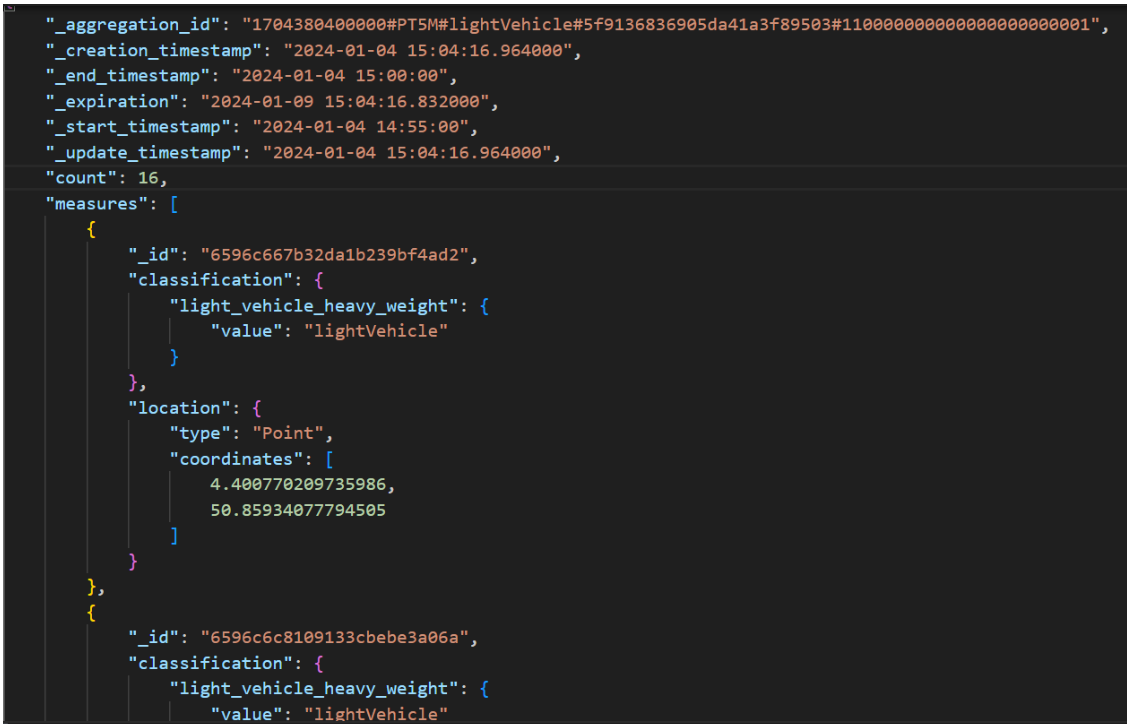


Figure 3.17: Transfer process of ANPR data



```
    "_aggregation_id": "1704380700000#PT5M#heavyWeight#LU#5ff8732eaa97ec0c71e0f55c#110000000000000000000001",
    "_creation_timestamp": "2024-01-04 15:05:28.424000",
    "_end_timestamp": "2024-01-04 15:05:00",
    "_expiration": "2024-01-09 15:05:28.182000",
    "_start_timestamp": "2024-01-04 15:00:00",
    "_type": "counting",
    "_update_timestamp": "2024-01-04 15:05:28.424000",
    "count": 1,
    "location": {
        "coordinates": [
            4.367423804739459,
            50.842784602836275
        ],
        "type": "Point"
    },
    "nationality": "LU",
    "period": "PT5M",
    "source": {
        "name": {
            "fr": "Rue Belliard - Avenue des Arts 46 2- Site Survey 34139",
            "vi": "Rue Belliard - Avenue des Arts 46 2- Site Survey 34139",
            "nl": "Rue Belliard - Avenue des Arts 46 2- Site Survey 34139",
            "de": "Rue Belliard - Avenue des Arts 46 2- Site Survey 34139",
            "nl-BE": "Rue Belliard - Avenue des Arts 46 2- Site Survey 34139",
            "en": "Rue Belliard - Avenue des Arts 46 2- Site Survey 34139"
        },
```

Figure 3.18: Example of counting data



```
    "_aggregation_id": "1704380400000#PT5M#lightVehicle#5f9136836905da41a3f89503#110000000000000000000001",
    "_creation_timestamp": "2024-01-04 15:04:16.964000",
    "_end_timestamp": "2024-01-04 15:00:00",
    "_expiration": "2024-01-09 15:04:16.832000",
    "_start_timestamp": "2024-01-04 14:55:00",
    "_update_timestamp": "2024-01-04 15:04:16.964000",
    "count": 16,
    "measures": [
        {
            "_id": "6596c667b32da1b239bf4ad2",
            "classification": {
                "light_vehicle_heavy_weight": {
                    "value": "lightVehicle"
                }
            },
            "location": {
                "type": "Point",
                "coordinates": [
                    4.400770209735986,
                    50.85934077794505
                ]
            }
        },
        {
            "_id": "6596c6c8109133cbebe3a06a",
            "classification": {
                "light_vehicle_heavy_weight": {
                    "value": "lightVehicle"
```

Figure 3.19: Example of speed statistic data

Chapter 4

Added Values and Use Cases

In this chapter we will go through the results of all the implementations that have been done or have been analyzed in order to allow the futures persons that will continue this work to have a clear vision of what is expected from them. There is 5 majors path to follow :

- **Visualization tool**
- **STIB alarms**
- **STIB trip**

each of the works will be explained more in details in the respective section. We will discuss about the current state of the work, why it is complicated to go further and what could be added to improve the work.

4.1 STIB Trips

The main added value of this work is the possibility to construct the trip for any vehicle of the STIB that is actually in service and is sharing his data through Open Data Portal of the STIB. The features that will be described in next Sections are all based on the this one. In the scope of this work only two use cases have been developed but since we can construct MobilityDB trips we can use all the other functions that MobilityDB can propose.

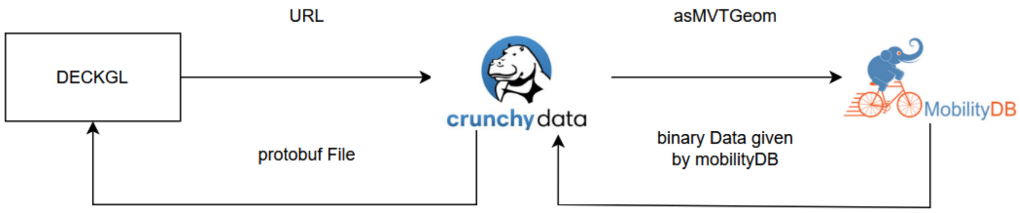


Figure 4.1: Communication process to get the visualization

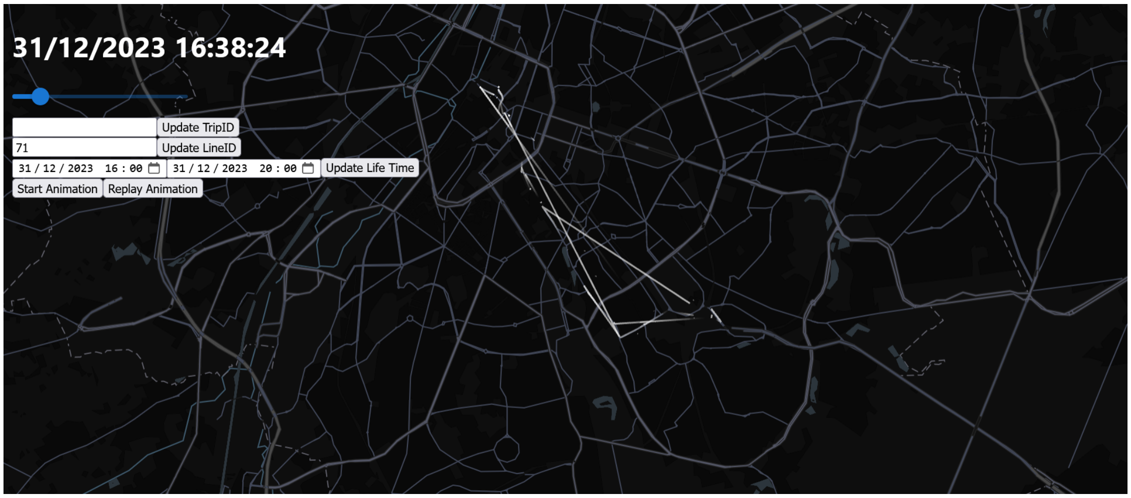
4.2 Visualization Tool

In this section we will discuss about the visualization tool that has been implemented in order to visualize the data provided by the STIB through the ODP, and show the two visualisation tool that compose the main tool. The two part that compose it are one base on the visualization of historical data allowing to chose the moment that the user is interested to watch from the past and a tool to see the position of every vehicle in real-time. Both tools are written using the framework DeckGL and the technology of Mapbox Vector Tile (MVT) to allow better performances as explained in Section 2.3 and the usage of pg_tileserv (that was presented in Section 2.4) as shown in figure 4.1 .

4.2.1 Historical Data Visualization

The module under review in this segment is architected to streamline the process of examining vehicle dynamics within the STIB network historically. It confers upon users the capacity to designate specific temporal brackets for analysis, facilitating the observation of vehicle or line behaviors within those defined epochs, conditional upon data congruence for the period in question.

Foremost, the module's functionalities bifurcate into two principal visualization modalities: the surveillance of individual vehicular movements and the aggregate tracing of line-wide transit patterns. Such a dichotomy permits an in-depth exploration of singular trajectories as well as comprehensive views of the network's flux. Nonetheless, it is crucial to acknowledge the granularity limitations of historical data. At times, these constraints precipitate the emergence of anomalous or implausible visual depictions, as exemplified in figure 4.2.

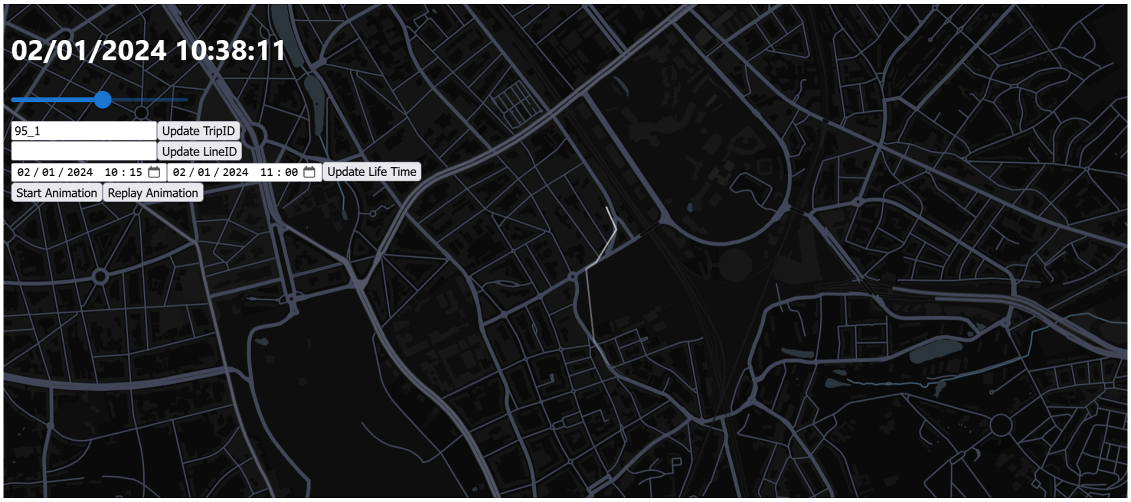


31/12/2023 16:38:24

71

31 / 12 / 2023 16 : 00 — 31 / 12 / 2023 20 : 00

Figure 4.2: Displayed data make no sense



02/01/2024 10:38:11

95_1

02 / 01 / 2024 10 : 15 — 02 / 01 / 2024 11 : 00

Figure 4.3: Displayed data make no sense

The visual apparatus, as depicted in Figure 4.2, is equipped with interactive mapping, temporal navigation, and animation controls. These instruments authorize the user to dynamically alter the visualization parameters, encompassing trip ID, line ID, and the data's lifecycle. While the visualization interface is meticulously engineered to yield an intuitive and educational exploration of the network's historical vehicle patterns, it must be conceded that data fidelity issues can sporadically induce atypical graphical representations. Despite these sporadic anomalies, the interface endeavors to furnish users with a coherent and insightful historical data visualization experience, consolidating its utility in the realm of urban transport network analysis.

When the underlying database retains data integrity, the module's visualization capabilities are excellent as we can see in Figure 4.3, highlighting its robust potential. Accurate data serves as the basis for the generation of consistent and precise visual narratives, enabling the module to vividly reconstruct the historical movements of vehicles on the network with remarkable fidelity. In cases where data is verified and reliable, visual outputs are not only accurate but also intuitive, facilitating an immersive analytical experience. Graphical renderings, in such cases, are seamless, providing a lucid and continuous representation of vehicle trajectories over time. This high-fidelity visualization is testament to the tool's sophisticated design and implementation, affirming its value as an essential asset for in-depth transit network analysis.

As explained in the previous section this result has been achieved by merging the power of multiples tools and the fact that pg_tileserv can handle requests for multiple layers at the same time. For every layer tht we would like to display on the screen there is an associated function layer written in postgresql and to which pg_tileserv can acces and run to get the protibuf data that will be provided to the MVTLayer of DeckGL. Both function layers for the single vehicle visuliation and all the vehicles of the line are showed in 4.1 and 4.2. The forced conversion of the trips from SRID 4326 to SRID 3857 is due to the fact that pg_tileserve default SRID is set to 3857 and proving data with different SRID make the resulting points to not be display at the right position because of the difference between every SRID has is own way to represent coordinates on a map.

Listing 4.1: Query to generate the protobuf data for one vehicle

```
CREATE OR REPLACE FUNCTION public.historical_trip(
z integer, x integer, y integer, p_tripid text, p_start text, p_end text)
RETURNS bytea
AS $$
WITH bounds AS (
   SELECT ST_TileEnvelope(z,x,y) as geom
   ),
   trips AS (
   SELECT asMVTGeom(transform(attime(trip,span(p_start::timestamptz,
      p_end::timestamptz, true, true)),3857),
   transform((bounds.geom)::stbox,3857)) as geom_times
   FROM stib_trips, bounds
   WHERE tripid = p_tripid
   ),
  mvtgeom AS (
  SELECT (geom_times).geom, (geom_times).times
  FROM trips
  )
SELECT ST_AsMVT(mvtgeom)
FROM mvtgeom
$$
LANGUAGE 'sql'
STABLE
PARALLEL SAFE;
```

Listing 4.2: Query to generate the protobuf data for all the vehicles of the line

```
CREATE OR REPLACE FUNCTION public.historical_line(
z integer, x integer, y integer, p_lineid text, p_start text, p_end text)
RETURNS bytea
AS $$
WITH bounds AS (
SELECT ST_TileEnvelope(z,x,y) as geom
),
trips AS (
SELECT tripid,asMVTGeom(transform(attime(trip,span(p_start::timestamptz, p_end::timestamptz, true, true)),3857),
transform((bounds.geom)::stbox,3857))
as geom_times
FROM stib_trips, bounds
WHERE lineid = p_lineid
),
mvtgeom AS (
SELECT (geom_times).geom, (geom_times).times,tripid
FROM trips
)
SELECT ST_AsMVT(mvtgeom)
FROM mvtgeom
$$
LANGUAGE 'sql'
STABLE
PARALLEL SAFE;
```

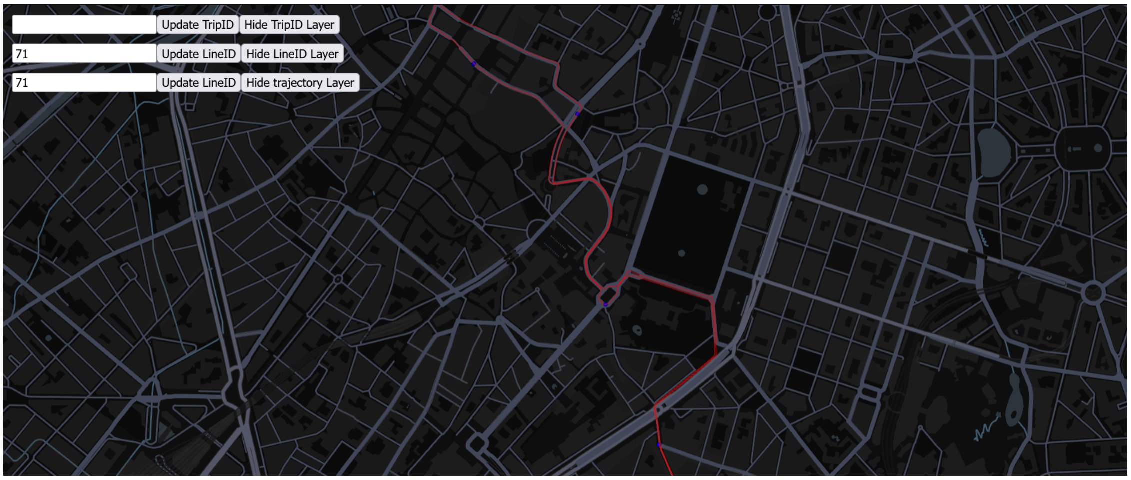


Figure 4.4: Real-Time position of some vehicle of line 71

4.2.2 Real-Time visualization

The visualization module delineated in this section epitomizes a considerable stride in the real-time portrayal of geospatial transit data. Capitalizing on the sophisticated rendering capabilities of deck.gl and the efficiency of Mapbox Vector Tiles (MVT), this tool is pivotal in providing actionable insights into the Société des Transports Intercommunaux de Bruxelles (STIB) network operations. Through the graphic representation of vehicle positions—marked as nodes—on the network map, the system affords a detailed snapshot of the real-time movement of each vehicle, facilitating the discernment of transit patterns and vehicular flow.

The provisioned Figure 4.4underscores the module's ability to trace individual vehicle trajectories, which are dynamically updated at intervals not exceeding 30 seconds. This frequent data refreshment ensures that stakeholders are equipped with the most current information, thereby supporting timely decision-making processes. Furthermore, the user interface is designed to enable focused examination of a single vehicle's route or, alternatively, an aggregated perspective of all vehicles on a selected line. Such a feature is instrumental for both immediate operational assessment and longitudinal data analysis, catering to varied analytical requirements.

While deck.gl provides the robust data visualization infrastructure, the use of MVT facilitates the handling of extensive datasets with minimal performance impact, thus ensuring the tool's responsiveness and scalability. The interactive options presented, including 'Update TripID' and 'Hide TripID Layer', offer users a degree of customization in their interaction with the visualization, further extending its utility across multiple urban mobility research contexts.

In conclusion, the visualization module presented here integrates advanced geospatial technologies to deliver a functional and responsive tool for urban transport visualization. It successfully furnishes a near-real-time overview of vehicle positions within the STIB network, albeit with the understanding that the fast-paced nature of urban transport may present challenges that this tool can only partially address. As such, while it stands as a valuable contribution to the field of urban mobility, the inherent limitations of real-time data analysis and the potential for information lag must be acknowledged.

As for the previous section in order to display the data on the screen a function layer has been written for every options that the users would display or not. The function are showed in 4.3 for the function that allow to follow one vehicle in real-time, in 4.4 for the function that allow to follow all the vehicles of one line in real-time, and 4.5 for the function that allow to display the entire trajectory that the vehicle is supposed to follow.

Listing 4.3: Query to get the last position of the desired vehicle

```sql
CREATE OR REPLACE FUNCTION public.trip_last_instant(z integer, x integer,
    y integer, p_tripid text)
RETURNS bytea
AS $$
WITH bounds AS (
SELECT ST_TileEnvelope(z, x, y) AS geom
),
last_instant AS (
SELECT tripid,asMVTGeom(transform(endinstant(trip), 3857),
    (bounds.geom)::stbox) AS geom_times
FROM stib_trips, bounds
WHERE tripid = p_tripid and current = true -- Filter by 'current' column
),
mvtgeom AS (
SELECT (geom_times).geom, (geom_times).times
FROM last_instant
)
SELECT ST_AsMVT(mvtgeom)
FROM mvtgeom
```

```sql
$$
LANGUAGE 'sql'
STABLE
PARALLEL SAFE;
```

Listing 4.4: Query to get the last position of the vehicles of one line

```sql
CREATE OR REPLACE FUNCTION public.line_last_instant(z integer, x integer, y integer, p_lineid text)
RETURNS bytea
AS $$
WITH bounds AS (
SELECT ST_TileEnvelope(z, x, y) AS geom
),
last_instant AS (
SELECT asMVTGeom(transform(endinstant(trip), 3857), (bounds.geom)::stbox) AS geom_times
FROM stib_trips, bounds
WHERE lineid = p_lineid and current = true -- Filter by 'current' column
),
mvtgeom AS (
SELECT (geom_times).geom, (geom_times).times
FROM last_instant
)
SELECT ST_AsMVT(mvtgeom)
FROM mvtgeom
$$
LANGUAGE 'sql'
STABLE
PARALLEL SAFE;
```

Listing 4.5: Query to the trajectory of the desired line

4.3 STIB Alarms

In this section we will discuss about the alarm system that has been created in order to help BM detect vehicles that are likely blocked on their path and that can create a lot of troubles in the traffic fluidity which is the most important thing for BM and why it is unfortunately for the moment not enough precise to consider all the alarms as correct.

The catalog of error messages communicated to Brussels Mobility (BM) plays a pivotal role in the optimization of traffic flow. To achieve the intended outcome, we have employed the stops function from MobilityDB. The query used in order to achieve the result is 4.6.

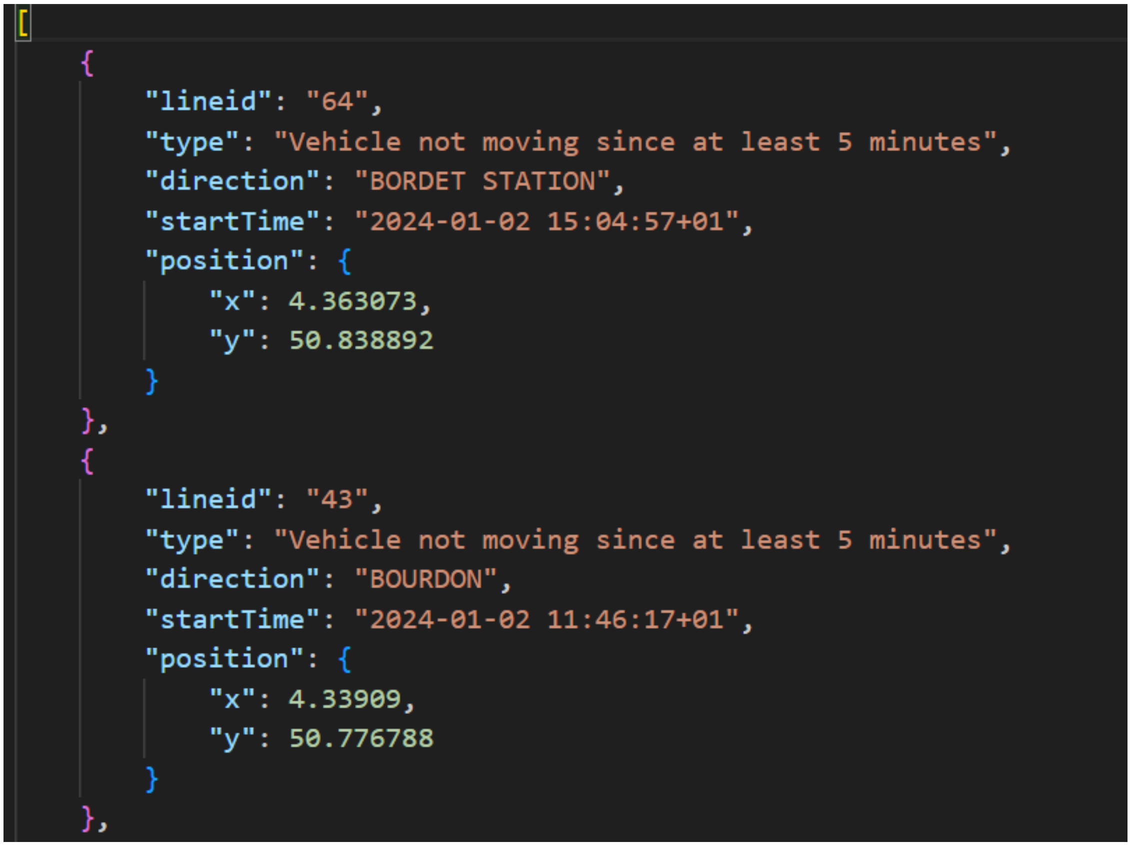
Listing 4.6: Query to generate the alarm for every vehicle that is currently onthe network

```
select lineid,trip_headsign,astext(unnest(sequences(stops(trip,5,'5
   minutes'))))
from stib_trips as st, terminus_shapes as ts
where tempSubType(trip) != 'Instant' and current = true and
   ts.route_short_name = st.lineid
group by trip,lineid,trip_headsign;
```

This function is instrumental in analyzing MobilityDB trip data to pinpoint instances where a vehicle remains within a confined spatial radius—defined by a predetermined size—for a minimum duration threshold. In this context, our parameters were set to identify vehicular stagnation exceeding a 5-meter radius for a duration of no less than 5 minutes. Upon detection of such an event, our system triggers an alarm that is subsequently encoded into a JSON file. This file is then dispatched to the relevant operational personnel for further examination and intervention as deemed necessary. This procedure is integral to the proactive management of traffic and the mitigation of potential congestion.

The output of the file is given by the image **??**, we can see that for every alarm there is multiples fields that can help the operator to understand where and since when the problem is occurring.

Because of the problems mentioned in the previous Sections all the alarms are not 100% reliable, but as soon as all the vehicles of the STIB will have an unique identifier for every vehicles all the features already implemented will be 100% functional.



```
[
    {
        "lineid": "64",
        "type": "Vehicle not moving since at least 5 minutes",
        "direction": "BORDET STATION",
        "startTime": "2024-01-02 15:04:57+01",
        "position": {
            "x": 4.363073,
            "y": 50.838892
        }
    },
    {
        "lineid": "43",
        "type": "Vehicle not moving since at least 5 minutes",
        "direction": "BOURDON",
        "startTime": "2024-01-02 11:46:17+01",
        "position": {
            "x": 4.33909,
            "y": 50.776788
        }
    },
```

Figure 4.5: Example of alarm file

Chapter 5

Future work

This study has allowed MobilityDB to start investing the path to be one of the main tool to handle the traffic in the city of Brussels. Nevertheless there is still a lot of possibilities to explorer that can lead to the implementation of new functionalities in MobilityDB that would make this extension even more efficient.

5.1 STIB Trips

The feature presented here represents the crux of this study and will be instrumental in shaping future work. Discussions are underway between BM and STIB to ascertain a timeline for the enhancement of STIB's real-time positioning system, with improvements projected to be completed by 2025. To refine the current system, it is necessary to devise an intricate algorithm that preserves each incoming data batch for comparison with its predecessor. This process will involve numerous interactions with the database to accurately identify consistent vehicle trajectories between successive batches.The challenge lies in the brief 20-second intervals between batch arrivals, which complicates the task of processing the entirety of the network's vehicular data. Currently, the system manages to update all trip details within an average span of 4 seconds during midday peaks. Incorporating these extensive computations is likely to extend processing times beyond the 20-second mark, potentially leading to data omission. Furthermore, while this proposed solution aims to minimize data inaccuracies, it acknowledges the inevitability of residual errors within the datasets.

5.2 STIB Alarms

The STIB alarm system is slated for an upgrade to align with the expected improvements to STIB's real-time positioning system by 2025. This enhancement is crucial for efficiently identifying and addressing service disruptions that could impact traffic flow and service reliability.

In case we would augment the current system alarm before 2025 a focus will be placed on refining the method of trip construction. This will involve the implementation of an improved approach to process and analyze incoming data more accurately. The aim is to enhance the alarm system's ability to track vehicles and detect anomalies more precisely, thereby reducing false positives and increasing overall system reliability.

A key challenge in this upgrade is to manage the increased data processing requirements within the limited time frame available between data batches. The current system updates trip information in about 4 seconds during peak times, and it is vital to ensure that the enhanced system does not exceed this update interval. Efficient processing is essential to prevent data loss and maintain the system's effectiveness.

This upgrade of the STIB alarm system is a critical aspect of our future work. It represents an opportunity to showcase the capabilities of MobilityDB and its application in a real-world, high-impact scenario. The successful refinement of this system will be a significant step forward in advancing urban traffic management solutions.

5.3 Vehicle Counting (M3)

The work of associating M3 count data with MobilityDB is independent of what was presented in the previous sections. There are still many possibilities to be explored, but a first avenue to explore would be to use MobilityDB's tint type to create a history of count data that would firstly be stored in a more efficient way by removing all the superfluous data we saw in Section 3.5, but which would also allow visualization over time just as has been done with vehicle trip data from the STIB network. For the visualization, we could imagine a circle that increases or decreases in size around the point where the camera is located over time, to show the density of vehicles around the camera.

5.4 Traffic Lights States (TLC)

As with the data in the previous section, no real results could be obtained from the data provided by TLC. However, it would be very interesting to be able to look into this, knowing that red light data can provide a great deal of insight into the traffic situation at any given moment, and that being able to manage red light behavior in a city can greatly improve traffic flow at critical points during rush hour, or even help understand what happened at a point where there was a problem with traffic flow.

5.5 Visualization Tool

As explained in the previous sections this implementation depends from the implementation of the STIB trips and the analyses of other data sources to which BM could have the access to. Nevertheless we can imagine additional features depending on the user requirements. The possibilities of information that we can display from the static GTFS data and the constructed MobilityDB trips are numerous but a good knowledge of the problems that are related to traffic flow is necessary in order to develop the right visualization.

We can imagine an improvement of the tool as being the addition of the visualization of counted vehicles around an area and the state of the traffic lights and then replay the situation and analyze from where was the problem coming, was it a vehicle blocking the road or to much vehicles in the area by looking at the counting or the traffic light behavior that wasn't adapted to the flow at that moment and a lot of other scenarios like that.

Chapter 6

Conclusion

This study has illuminated the transformative capabilities of MobilityDB for enhancing urban traffic management in Brussels. Through the integration and analysis of real-time and historical data from STIB's network, we have established a foundational framework that sets the stage for a transport system that is both intelligent and adaptable.

The implementation of the features discussed herein not only serves immediate operational needs but also opens a myriad of possibilities for MobilityDB. This project has presented an invaluable opportunity to demonstrate the full potential of this extension. Its success lays testament to the versatility and strength of MobilityDB in handling complex spatiotemporal data.

The current phase of our research has been primarily devoted to solidifying reliable data streams and crafting initial analytical tools. However, the journey ahead is filled with potential. The forthcoming enhancements to STIB's real-time positioning system, expected to be complete by 2025, will play a critical role. Moreover, the sophisticated algorithm planned for implementation aims to further refine the accuracy of data processing. It is important to bear in mind, nonetheless, that some level of imperfection is inherent in real-time data analytics.

In sum, the advancements made thus far are significant and indicative of a positive trajectory. Yet, the pursuit of a fully optimized, data-driven transit system for Brussels remains a work in progress. The sustained pursuit of innovation, together with upcoming improvements in STIB's data infrastructure, will undoubtedly enable more efficient and dependable urban transportation solutions. Through these efforts, MobilityDB stands to not only prove its worth but also to carve a niche as an indispensable tool in the realm of geographic information systems.