JACOBS
UNIVERSITY

Syed, Nizam Uddin

Roadlytics: a digital solution to minimize traffic congestion in the city of Hamburg.

Master Thesis
May 2021

Data Engineering Graduate Program

Student Credentials:

- Syed, Nizam Uddin
- Matriculation number: 30002112
- Master's Thesis

English: Declaration of Authorship

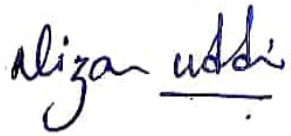
I hereby declare that the thesis submitted was created and written solely by myself without any external support. Any sources, direct or indirect, are marked as such. I am aware of the fact that the contents of the thesis in digital form may be revised with regard to usage of unauthorized aid as well as whether the whole or parts of it may be identified as plagiarism. I do agree my work to be entered into a database for it to be compared with existing sources, where it will remain in order to enable further comparisons with future theses. This does not grant any rights of reproduction and usage, however.

This document was neither presented to any other examination board nor has it been published.

German: Erklärung der Autorenschaft (Urheberschaft)

Ich erkläre hiermit, dass die vorliegende Arbeit ohne fremde Hilfe ausschließlich von mir erstellt und geschrieben worden ist. Jedwede verwendeten Quellen, direkter oder indirekter Art, sind als solche kenntlich gemacht worden. Mir ist die Tatsache bewusst, dass der Inhalt der Thesis in digitaler Form geprüft werden kann im Hinblick darauf, ob es sich ganz oder in Teilen um ein Plagiat handelt. Ich bin damit einverstanden, dass meine Arbeit in einer Datenbank eingegeben werden kann, um mit bereits bestehenden Quellen verglichen zu werden und dort auch verbleibt, um mit zukünftigen Arbeiten verglichen werden zu können. Dies berechtigt jedoch nicht zur Verwendung oder Vervielfältigung.

Diese Arbeit wurde noch keiner anderen Prüfungsbehörde vorgelegt noch wurde sie bisher veröffentlicht.



Signature
Syed, Nizam Uddin.

Place, Date
Bremen, 17 May 2021

Roadlytics: a digital solution to minimize traffic congestion in the city of Hamburg.

A novel approach for developing a tracking device, collecting Geo-referenced data and solving congestion problem using AI/ML

Syed, Nizam Uddin

Data Engineering Graduate School
Jacobs University Bremen gGmbH
Campus Ring 12
28759 Bremen
Germany

E-Mail: n.syed@jacobs-university.de
http://www.jacobs-university.de/

Abstract

The aim of this project is to develop a portable, low-cost vehicle tracking device based on Internet of Things (IOT) technology for the purpose of collecting precise and accurate Geo-location data in order to minimize traffic congestion problem caused by logistic vehicles by applying Machine Learning (ML) and Artificial Intelligence (AI). Additionally, this data will be used in conjunction with other sources to built Smart Delivery and Loading Zones (SmaLa) a mobility project by Behörde für Verkehr und Mobilitätswende (BVM) Hamburg. The complete project is built in Landesbetrieb Straßen, Brücken und Gewässer (LSBG) as an in-house development project called Roadlytics.

LSBG is a service provider for Hamburg administration with main focus on Urban Development, City Planning and Intelligent Transport Systems (ITS). LSBG is in the process of generating different data-sets related to mobility sector as part of smart city initiative, however currently they don't have the specific data what we are producing in this project instead they are applying legacy approach of counting vehicles manually and forecasting congestion's in the city.

This project identifies the shortcomings of current scenario and proposes better way to collect and analyse the data for the city planners and other departments. We showcase that it is possible to solve the congestion issue by implementing using state of the art methods. We also make this data available to citizens on Urban Data Platform (UDP) Hamburg.

Acknowledgements

I would first like to express my heartfelt appreciation and sincere gratitude to everyone who assisted me in any way during my thesis period. I am extremely grateful to my thesis supervisor **Dr. Fangning Hu**, whose constant motivation, relentless inspiration, strong encouragement, and excellent guidance enabled me to complete my thesis successfully.

I would also like to thank **Dr. Meleanie Mergler**, project manager #transmove and GeoPro at LSBG Hamburg, for all her assistance and guidance as my second thesis supervisor.

Finally, I am grateful to my bosses **Mr. René Binnewerg** and **Miss. Marina Zöfeld** for their extreme support, and trusting my skills and giving me an opportunity to explore all the available resources in DigiLab, LSBG; Last but not the least, I am very much thankful to my parents, family members, my girlfriend and friends, who provided a strong mental support and continuous encouragement to complete my work on time. I would like to thank everyone again for their contributions to my success. Thank you so much

Thesis Supervisors and Reviewers

Primary Supervisor	**Dr. Fangning Hu** **University Lecturer** **Computer Science & Electrical Engineering** **Jacobs University Bremen gGmbH,** **Campus Ring 1, D-28759 Bremen (Germany)** **Tel: +49 421 200-3515** **Fax: +49 421 200-3103** **Email: f.hu@jacobs-university.de** **Homepage:** **http://fhu.user.jacobs-university.de** **Office: Research I, Room 92**
Secondary Supervisor	**Dr. Melanie Mergler** **Georeferenzierte Projekte (GeoPro)** **Projektleitung #transmove** **Freie und Hansestadt Hamburg** **Landesbetrieb Straßen, Brücken und Gewässer** **Sachsenfeld 3-5 • 20097 Hamburg** **Telefon +49 40 428 26-2806** **Mobil +49 151 232 734 57** **Email melanie.mergler@lsbg.hamburg.de** **www.lsbg.hamburg.de**
First Reviewer	**René Binnewerg** **Leitung Digitales Labor – DS2** **Programmleitung Georeferenzierte Projekte (GeoPro)** **Freie und Hansestadt Hamburg** **Landesbetrieb Straßen, Brücken und Gewässer** **Sachsenfeld 3-5 • 20097 Hamburg** **Telefon +49 40 428 26-23 46** **Mobil +49 176 428 60-223** **E-Mail : rene.binnewerg@lsbg.hamburg.de**
Second Reviewer	**Marina Zöfeld** **Leitung Digitales Labor – DS2** **Programmleitung Georeferenzierte Projekte (GeoPro)** **Projektleitung GeoNetBake und stellv. Projektleitung DigITAll** **Freie und Hansestadt Hamburg** **Landesbetrieb Straßen, Brücken und Gewässer** **Sachsenfeld 3-5 • 20097 Hamburg** **Telefon +49 40 428 26-21 41** **Mobil +49 176 428 60-227** **E-Mail marina.zoefeld@lsbg.hamburg.de**

Contents

1 Introduction **1**
1.1 Introduction . . . 1
1.1.1 Problem Statement . . . 3
1.1.2 Smart Cities . . . 4
1.1.3 Internet of Things . . . 5

2 Background and Approach **6**
2.1 Problem space exploration . . . 7
2.2 Problem Validation . . . 7
2.3 Solution Ideation . . . 9
2.4 Defining the solution in a nutshell . . . 9

3 Understanding Geospatial data **11**
3.1 GPS and GNSS data . . . 11
3.1.1 Working of GPS . . . 11
3.1.2 Difference between GPS and GNSS . . . 13
3.2 Georefernced data from GNSS receivers in the form of raw data . . . 14
3.2.1 Decoding single NMEA sentence . . . 14
3.3 Complete workflow of GNSS trackers in Roadlytics Application . . . 16

4 Hardware Prototypes **18**
4.1 Prototype Version 1 - GNSS, Raspberry Pi 4 [GNSS Data Logger] . . . 18
4.1.1 Hardware Requirements List . . . 18
4.1.2 Hardware Configuration and setup . . . 22
4.1.3 Assembling Hardware Modules and Setting up Software . . . 26
4.1.4 Data Processing and Visualization . . . 29
4.2 Prototype Version 2 - GNSS module, Raspberry Pi ZERO Wireless [Upgraded GNSS Data Logger] . . . 30
4.2.1 Hardware Requirements List . . . 30
4.2.2 Working Prototype Version 2 and it's Software Stack . . . 30
4.2.3 Data Processing and Visualization . . . 31
4.3 Prototype Version 3 - GNSS, raspberry pi zero wireless and pi battery hat with initial state IOT platform . . . 31
4.3.1 Hardware Requirements List . . . 31
4.3.2 IoT platform - Initial state . . . 33

4.4 Prototype Version 4 - RAK7200 raspberry pi 4 and RAK2245 . . . 35
4.4.1 Hardware Requirements list . . . 37
4.5 Prototype Version 5 - Final Industry Standard Production Ready Prototype . . . 48
4.5.1 This prototype is made up of the following hardware modules 49
4.5.2 Roadlytics IoT platform . . . 50
4.5.3 Roadlytics System Architecture . . . 53
4.5.4 Roadlytics Web App . . . 54

5 ML Implementation and Data Analysis 55
5.1 DBSCAN-***Density Based Spatial Clustering with Noise by sklearn*** . . . 55
5.1.1 Steps involved in DBSCAN Algorithm . . . 58
5.1.2 Evaluating Cluster performance . . . 58
5.1.3 DBSCAN Analysis on tracker dataset 1 . . . 59
5.1.4 DBSCAN Analysis on tracker dataset 2 . . . 62
5.2 Data Visualization . . . 65
5.2.1 Clustering hotspot points . . . 67
5.2.2 Displaying delivery routes on map . . . 69
5.2.3 Displaying data on map to track vehicle stop moments . . . 69
5.2.4 Displaying Van stopped for delivering . . . 69
5.2.5 Displaying data as heatmap . . . 69

6 Results and Conclusion 72
6.1 Results and Conclusion . . . 72
6.2 Future Work . . . 72

7 Appendix 73
7.1 Prototype Version 1 source code . . . 73
7.2 Prototype Version 3 source code - Initial State . . . 74
7.3 HTTP API - python flask framework . . . 75
7.4 Prototype Version 3 source code - Initial State . . . 77
7.5 Unsupervised ML model: DBSCAN implementation in python . . . 77

List of Figures

1.1 Smart City Ranking Index 2019 2
1.2 Hamburg's Population statistics 2019 3
1.3 Logistic vehicles parked on second row for delivering the parcels . . 3
1.4 Traffic Report Hamburg by TomTom 4
1.5 Smart City core elements 5

2.1 Design Thinking - double diamond framework picture [adapted: Digital Product School, Munich] 6
2.2 Design Thinking : List of pain points with respect to users/stakeholders 8
2.3 Lean Canvas 10
2.4 Roadlytics in a Nutshell 10

3.1 How GPS determines actual location of gps-receivers 12
3.2 GPS-track 13
3.3 GNSS-track 13
3.4 Tracker data comparison 13
3.5 GNSS raw data 14
3.6 Structure of NMEA protocol message 15
3.7 NMEA parser logic using python 16
3.8 GNSS tracker workflow in Roadlytics Application 17

4.1 Raspberry Pi 4 19
4.2 GNSS 4 click with integrated SAM-M8Q 20
4.3 Pinout Diagram 21
4.4 From Left: LiPo Powerbank, USB C Type Cable, SD Card, Jumper Wires 22
4.5 Writing Raspberry Pi OS on SD card using rpi-imager utility 23
4.6 Enabling SSH and network on boot 24
4.7 Enabling Serial Port 25
4.8 Enabling Serial Port 26
4.9 Prototype Version 1 27
4.10 Data generated form Serial Port 28
4.11 Architecture of Prototype 1 29
4.12 Raspberry PI ZERO W 31
4.13 Raspberry Pi zero measurement 32

4.14 Prototype Version 2 . . . 33
4.15 Li-Ion Battery Hat for Raspberry Pi . . . 34
4.16 Initial State account creation and login . . . 35
4.17 Bucket Creation . . . 36
4.18 Initial State Key generation . . . 36
4.19 Initial State data streaming and visualization . . . 37
4.20 Prototype version 3 - realtime GNSS/GPS tracker . . . 37
4.21 Prototype version 3 - dimension views . . . 38
4.22 LoRa Network Architecture . . . 38
4.23 RAK7200 Node . . . 40
4.24 RAK7200 device setup and configuration . . . 40
4.25 TTN Home Page and Console Page . . . 41
4.26 Adding an Application . . . 42
4.27 Registering and Adding Device . . . 43
4.28 Device Overview . . . 44
4.29 RAK2245 Pi Hat - LoRa Gateway Module . . . 44
4.30 Command line after log in to gateway . . . 46
4.31 LoRa Gateway ID . . . 46
4.32 LoRa Gateway Module Setup . . . 47
4.33 RAK2245 Pi Hat Gateway ID in SSH . . . 47
4.34 RAK2245 Pi Hat TTN Connection Success . . . 48
4.35 Final Paylaod in JSON fomrat . . . 48
4.36 Prototype Version 4 with LoRa . . . 49
4.37 Node Red Flow . . . 52
4.38 High level System Architecture of Roadlytics . . . 53
4.39 roadlytics WebApp Screenshots . . . 54

5.1 Understanding DBSCAN pictorially . . . 56
5.2 predicted hotspot from dbscan and plotted on folium map . . . 57
5.3 Model Performance $\Rightarrow SilhouetteScore(underfit, overfitandoptimalfit)$ 59
5.4 Test Case 1: eps = 0.0001 and minPts = 4 $\Rightarrow OptimalFit$ . . . 60
5.5 Test Case 1: eps = 0.0001 and minPts = 6 $\Rightarrow OptimalFit$ . . . 61
5.6 Test Case 1: eps = 0.0001 and minPts = 8 $\Rightarrow OptimalFit$ . . . 62
5.7 Test Case 1: eps = 0.0001 and minPts = 4 $\Rightarrow UnderFit$ . . . 63
5.8 Test Case 1: eps = 0.0005 and minPts = 6 $\Rightarrow OverFit$ . . . 64
5.9 Test Case 1: eps = 0.0001 and minPts = 6 $\Rightarrow optimalFit$ . . . 65
5.10 Map legends: Speed based on colors [low — Medium — High] . . . 66
5.11 Spotting Hotspots . . . 66
5.12 Density Near Traffic Signal Light . . . 67
5.13 Trip Data - 1 . . . 67
5.14 Hamburg Traffic Signal Data . . . 68
5.15 Trip data - 2 . . . 68
5.16 Visualization of 7 hotspots on folium maps . . . 69
5.17 Route taken by delivery vehicles - moving data . . . 70
5.18 Data with speed lessthan or equal to 5kmph . . . 70

5.19 Delivery vans stopped for delivering parcels where speed is zero(red dots are basically hotspots) 71
5.20 Delivery vans frequent routes as heatmap 71

List of Tables

3.1 Decoded '$GNRMC' sentence 15

4.1 PIN connection 27
4.2 Battery hat for Pi interface description 32
4.3 LED states when discharging 33
4.4 LED states when discharging 34
4.5 ZED-F9P performance in different GNSS modes 50

5.1 Comparison of dbscan results with different hyperparameter tuning on dataset 1 62
5.2 Comparison of dbscan results with different hyperparameter tuning on dataset 2 65

Acronyms

AI Artificial Intelligence. 3

COLD START First Time GNSS module power-on. 29

FCP Fast Charging Protocol Huawei™. 32

GNSS Global Navigational Satellite Systems. 6, 12, 13, 28

GPIO General Purpose Input/Output. 29, 45

GPS Global Positioning System. 6, 11, 13, 39

HOT START Subsequent Power On. 29

IOT Internet of Things. 2, 3, 5, 31, 39

JSON Javascript Object Notation. 29

KMPH Kilo Meter Per Hour. 65

LGV Landesbetrieb Geoinformationund Vermessung. 1

LoRa Long Range Network. 36, 37

LPWAN Low-power wide-area network modulation technique. 37

LSBG Landesbetrieb Straßen, Brücken und Gewässer. 2, 3

LTE-M Long-Term Evolution for Machine-Type Communications. 49

MCU Micro Controller Unit. 49

ML Machine Learning. 3

MQTT Message Queuing Telemetry Transport. 51

NMEA National Marine Electronics Association. 16, 29

PD Power Delivery. 32

PE Pump Express. 32

QC Quick Charge. 32

RF Radio Frequency. 39

RTK Real Time Kinematics. 50

SD Card Secure Digital Card. 45

SSH Secure Shell. 29

TTFF Time To First Fix. 13, 29

TTN The Things Network. 39, 44, 45

UART Universal Asynchronous Receiver/Transmitterr. 29

Chapter 1

Introduction

1.1 Introduction

Since the beginning of urbanization, mobility has always been a core concern and a central figure in cities development and success. In the past, people mainly related the term 'mobility' with the liberty to move around using new means of transportation. Nowadays, however, the term mobility is strongly linked with ***traffic congestion*** - an increasingly pressing issue for many large cities.The average time lost during the rush hours per 30 minutes trip is 16 minutes in the morning and 18 minutes in the evening, constituting to a loss of 131 hours a year per car user [1]. Hamburg is certainly among cities that are heavily impacted by traffic congestion. In fact, a statistics by `https://www.tomtom.com/de_de/` shows that Hamburg ranks first place for traffic jams in Germany.

Moreover, it is the second most populated city with 1.84 million inhabitants in 2019 see figure 1.2. According to a report from "Landesbetrieb Geoinformation- und Vermessung (LGV), Hamburg", by 2030 the population will increase by 100 thousand more [2]. This means a strong increase in number of vehicles on the road, which will again contribute to the city's traffic congestion, worsening the situation even further if no effective solutions are implemented.

So far many traffic planners have made numerous attempts to solve congestion related problem, mainly by introducing alternative means of transportation and improving city's routing. However, building new roads and bridges is very expensive, time consuming and not efficient enough to counteract the effects from growing number of vehicles. Being ranked as the "Smartest City" in Germany (for second consecutive years among 80 participating cities, according to 'Bitkom's Smart City Index report' [3]), Hamburg is now determined to use technology in their search for alternative solutions to traffic congestion. This is also where the concept of smart cities comes into play.

There are many causes for traffic congestion in Hamburg, namely:

- Construction sites
- Vehicles parked on second row
- **Delivery Van's Parking on side ways, pedestrian walkway and bike lane**
- Accidents
- Vehicle breakdown
- Slow moving traffic
- Drunk driving

Among the above listed reasons, side-way parking of logistic vehicles and their impact can be observed immediately on the streets of Hamburg, as shown in the figure 1.3, and it is also the issue of focus for this project by **Landesbetrieb Straßen, Brücken und Gewässer (LSBG), Hamburg**. As part of Hamburg "smart city" initiative, Roadlytics collects congestion related to delivery vehicles data and analysed using state-of-art IOT and AI/ML technology, in order to provide insights that will enable better planning for the city and minimise the traffic jams.

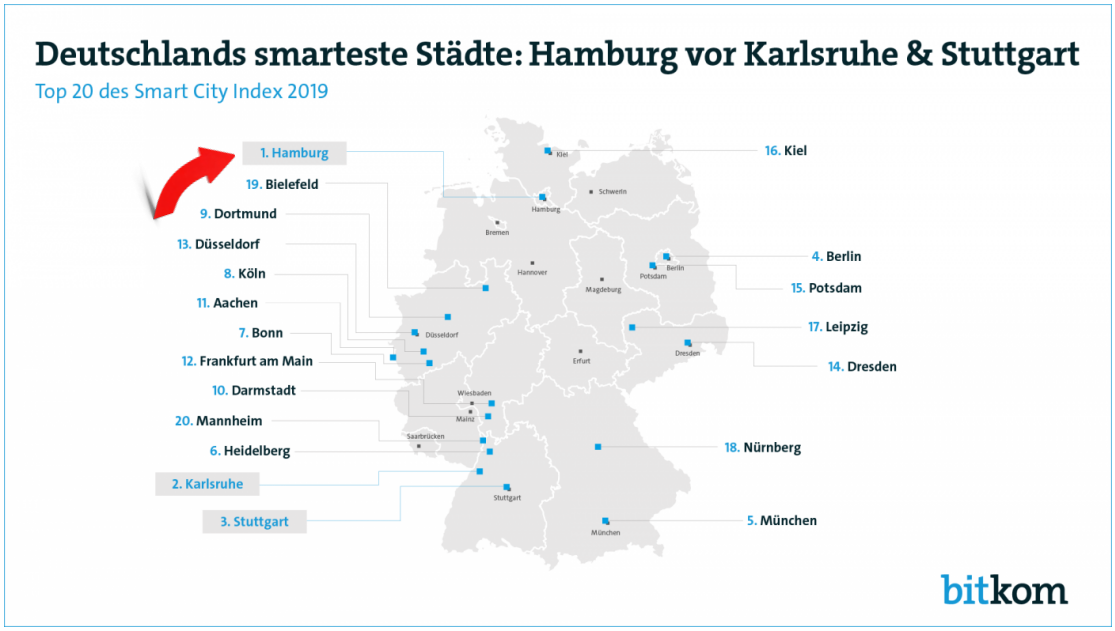


Figure 1.1: Smart City Ranking Index 2019
[adapted © Bitkom e.V]

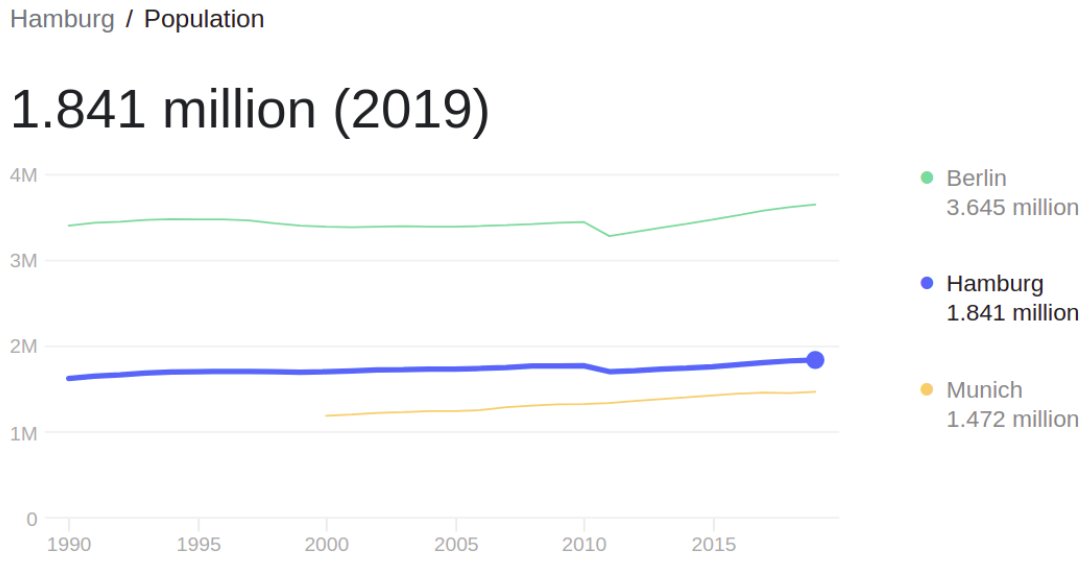


Figure 1.2: Hamburg's Population statistics 2019
[reproduced from Eurostat]

Figure 1.3: Logistic vehicles parked on second row for delivering the parcels
[source: self produced]

1.1.1 Problem Statement

Traffic congestion continues to be one of the biggest issues in various transport networks. The ever-increasing demand for mobility in urban areas has resulted

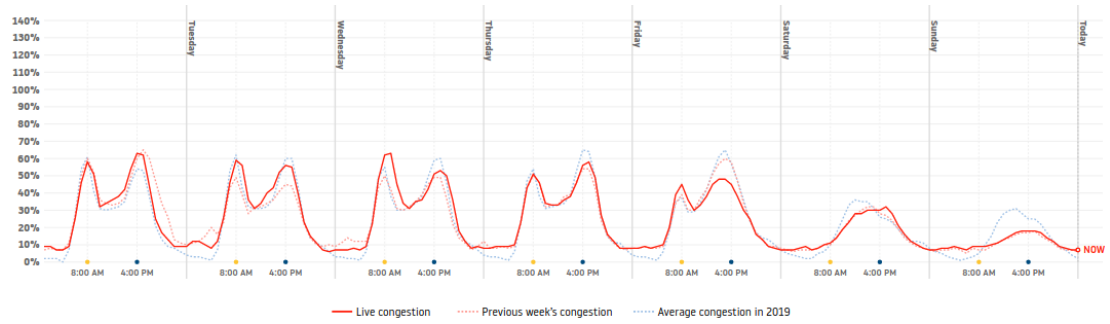


Figure 1.4: Traffic Report Hamburg by TomTom
[adapted from TomTom `https://www.tomtom.com/en_gb/traffic-index/hamburg-traffic`]

in increased traffic congestion and a multitude of problems associated with it. Logistics Van traffic is one of the the fastest growing sector of road traffic and the growth is still continuing. Both logistic companies and the city of Hamburg have a lack of information of delivery vans behavior in the city. Gaining a better understanding of how and why van traffic is growing would allow city planners to better grasp the issues and develop responses to make better decisions in creating effective solutions for both parties.

The issue we are attempting to address with this project is to collect data on delivery vehicles. To accomplish this, we are also prototyping low-cost commodity GNSS/GPS tracker capable of generating accurate geolocation data with a precision of less than a meter, finally we run AI/ML models on these data to predict which are the hotspots [4]

People are particularly worried about those parts of the city that are regularly overcrowded, as these areas could increase the impact and length of congestion. Previous awareness of these areas and a greater understanding of their effects will help to establish more effective traffic management strategies.

The successful implementation of this project would support the city of Hamburg by identifying the most frequent parking hotspots for delivery vans, analyzing the effects of traffic jams, planning new optimized routes, and providing stress-free parking for logistics vehicles.

1.1.2 Smart Cities

The concept of smart cities is to put **data and technology** to work in order to make better decisions and to improve quality of life, more comprehensively, real-time data gives an ability to watch events as they unfold, understand how demand pattern are changing and to respond with faster yet lower cost solutions. There are three important layers that makes a smart city.

1. First is the technology base, which includes a critical mass of smartphones and sensors connected by high-speed communication networks.
2. The second layer consists of specific applications. Translating raw data into alerts, insight, and action requires the right tools, and this is where technology providers and app developers come in.
3. The third layer is usage by cities, companies, and the public.

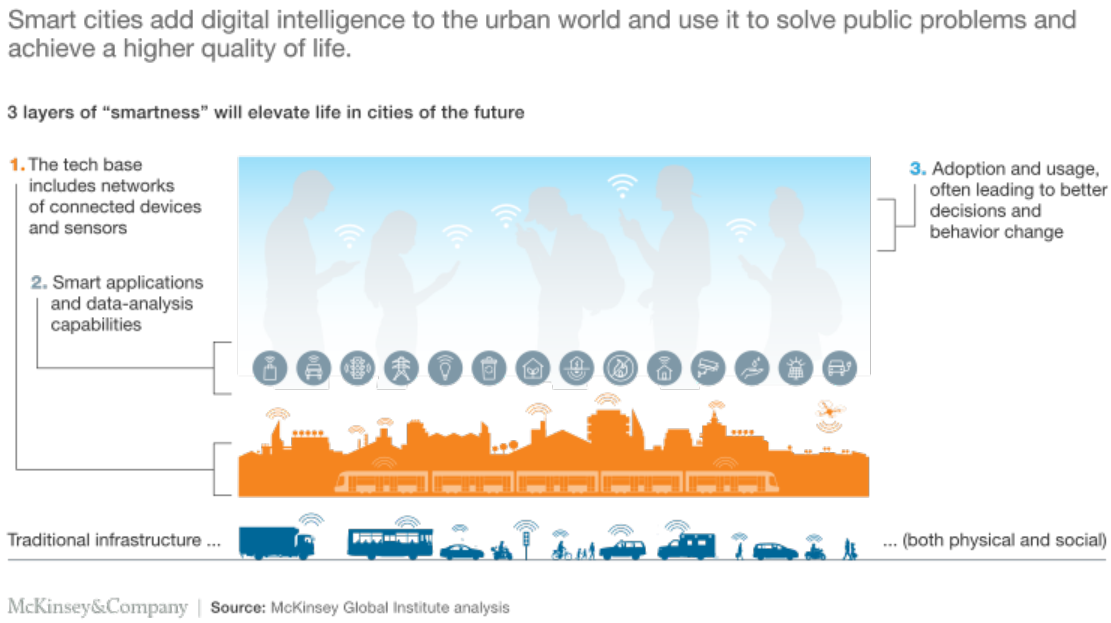


Figure 1.5: Smart City core elements
[reprinted © McKinsey&Company]

1.1.3 Internet of Things

Internet of Things (IOT) refers to the network of physical objects called as 'Things' that are embedded with sensors, software and other technologies for the purpose of connecting and transferring data with other devices and systems over the internet. IoT plays an important role in building smart cities [5], Hamburg has wide range of projects concerning to Intelligent Transportation System in the direction of urban mobility and logistics solutions, some of the running projects within LSBG alone are [6]

- LSA Plus - Traffic light forecasting
- Stauprognose - Traffic jam/congestion forecasting
- GeoNetBake - Sensor based live construction tracking
- DigitALL - Traffic related projects and ROADS - Construction planning tool

Chapter 2

Background and Approach

This section, we explain the problem background and approaches taken to solve the given problem statement as seen in the quotes below.

"Would it be great if parking violators and logistic vehicles did not disturb the traffic flow anymore?"

Based on this initial outset we went through several rounds of iteration of our product following the design thinking methods. Our final product was an IoT based GNSS/GPS-tracker and a web application that displays detailed information about delivery vans routes and delivery Hotspots. The goal of the product is to provide actionable data to city planners, enabling better planning and management of curb space.

Applying the double diamond as a framework [7], we divided the project roughly into four initial steps, problem space exploration, problem validation, solution ideation, defining the solution, followed by build, measure and learn iterations:

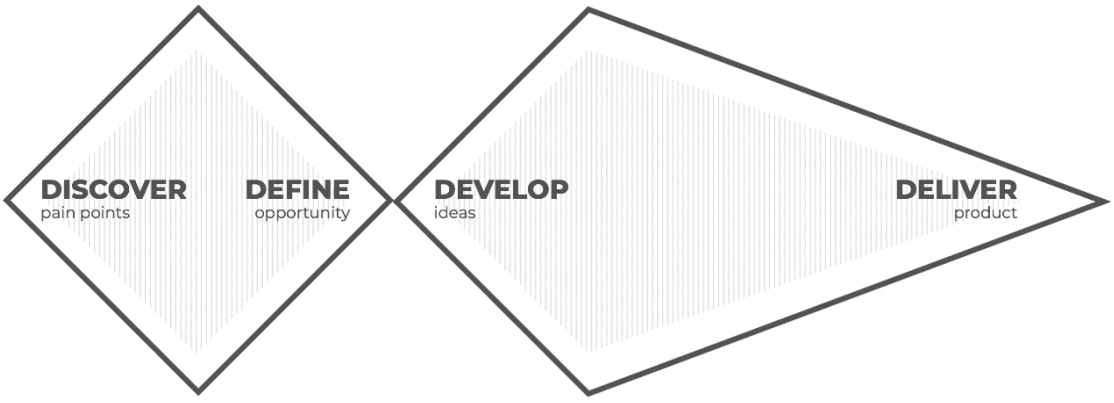


Figure 2.1: Design Thinking - double diamond framework picture [adapted: Digital Product School, Munich]

2.1 Problem space exploration

In the initial phase of the project, we spent approximately a month on research with the purpose of understanding different aspects of the problem, finding our target group and respective pain points and needs, defining hypotheses and validating them. Our investigation revealed that the curb area is becoming one of the most contested spaces in urban areas, demanding fast responses from authorities and attracting the attention of technology companies [8].

One of the main causes of the increasing traffic of delivery vehicles is the fact that the e-commerce market is expanding rapidly. During the period 2012 to 2019, online shopping grew by nearly 70%, and Germany is Europe's second largest online market, however it has the highest growth potential[9]. The exponential growth of online shopping for physical goods generates a significant demand for dedicated delivery services and results in increasingly difficult last mile logistics causing traffic issues within the urban areas[10]. Based on the above research, we recognize that the increase in van traffic is a phenomenon that has been growing rapidly and brings great challenges in terms of the operationalization of public spaces.

2.2 Problem Validation

In order to validate the problem more specifically, we conducted a series of interviews with our stakeholders and potential users. The goal was to identify specific pains and needs and list down the most relevant of them. As a start, we listed all possible users and stakeholders and clustered them in different categories such as public administration, private companies, individuals, etc. After that, we conducted user interviews and classified into three different groups. To avoid biasing the user answers, we formulated broad questions that would allow the interviewees to express themselves and expose their main concerns spontaneously. We clustered the pain points observed in the interviews according to the respective target groups in order to identify patterns. Based on the initial batch of interviews we identified the following pain points as stated in the chart 2.2

Based on this experience we re-defined our target groups and respective pains we decided to add **city planners** as one of the target groups based on the observation that they are one of the biggest players in this context. They not only are responsible for managing the use of public spaces, but also creating the structure to handle the issues concerning traffic flow and parking. Because of that, we decided to focus on two different users, city planners and traffic agents (agents responsible for law enforcement). This decision was made based on two main reasons:

- Innovation takes longer to be implemented in public departments than in other sectors resulting in significant technological gaps.

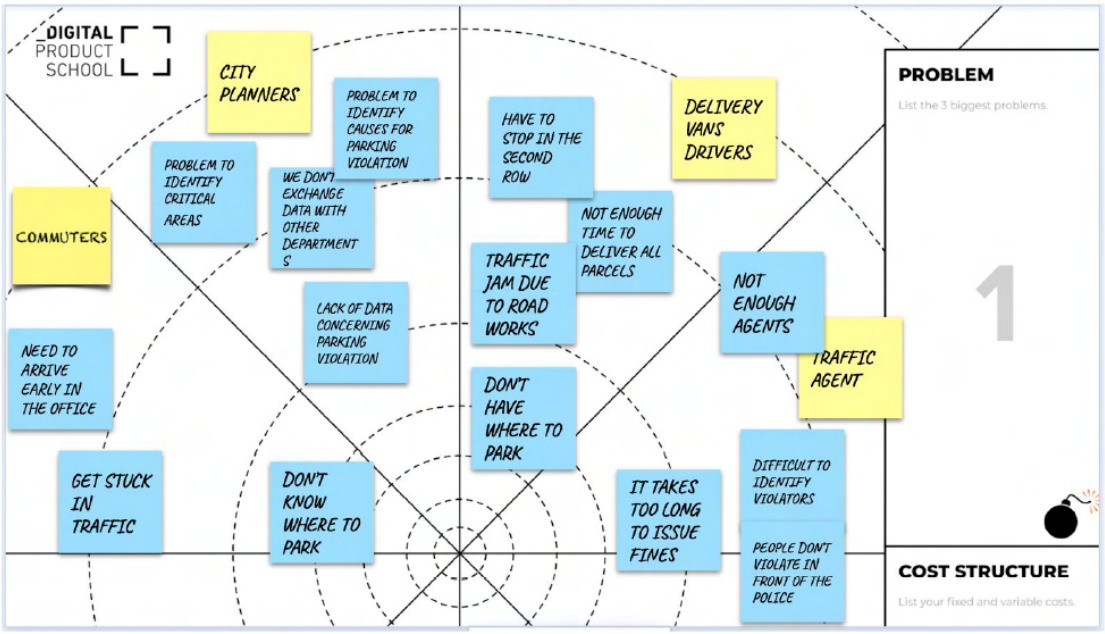


Figure 2.2: Design Thinking : List of pain points with respect to users/stakeholders [adapted from DPS Munich]

- Enabling improvements in public administration create a larger social impact affecting all segments involved in traffic.

Based on our research we identified that one of the biggest issues concerning delivery vans traffic, is irregular parking (second-row, bike lanes and sidewalks parking). It is known by the public administration that this is one of the factors generating traffic jams, but currently they don't have means to measure its impact. There is still a huge gap in terms of information, a better understanding of van traffic and parking patterns is needed to allow policymakers to better grasp the issues and develop responses.

During the interviews we realized that currently, most of the parking related data is collected manually, another part comes from the parking machines, but those just register the cars that are parked regularly. In this context, technology and data may offer opportunities to improve understanding of logistic van fleet usage. More research, data and analysis is needed to enable better planning, and consequently better execution of public policies. As mentioned earlier, we decided to focus on two different users firstly city planners and secondly logistic companies.

2.3 Solution Ideation

Formulating the first version of the Lean Canvas enabled us to summarize our ideas and create the structure for our next steps as depicted in the picture 2.3. At this point we had spotted our target groups and their pain points, next we clustered our findings and formulated our product vision: Providing tools to support parking optimization and better management of public resources. The next step would be to find a solution that would eliminate the user's obstacles and match with our vision. Filling the Canvas helped us to cut whatever was unnecessary, set our product goal and provided the basis for our solution ideation. After several rounds of brainstorming, we selected three options we believed would be a better fit for the user's pain points:

1. Applying AI/ML algorithm to analyze existing data to extract information about parking - We based this idea on the fact that there are already cameras installed in some critical points. With access to these videos, we could apply an algorithm to extract and store only the relevant information.

2. Installing sensors to collect data - For bureaucratic reasons we had to consider that we might not have access to the data already available. In this case we would have to generate our own data, and sensors are reliable and accurate sources.

3. Mobile app to identify and fine violators - We thought about that as a way to optimize the work of traffic agents at the same time that it would also generate data for the city planner.

Considering a variety of factors, we came to the conclusion that generating data through our own sensors would be the best option, this way we could select the most convenient and accurate data format also avoid bureaucratic procedures the get access to existing data.

2.4 Defining the solution in a nutshell

We will prototype GNSS-based tracker, will be placed in delivery vehicles to collect Geospatial data, we will then run AI/ML models on these datasets to capture hidden behaviour of the logistic vans and their parking hotspots, and the data will be displayed on a Roadlytics Web application. This allows city planners to design the city in a more efficient and convenient manner, and logistics companies to gain insights from these datasets.

Lean Canvas

The Problem	Solution	Unique Value Proposition	Unfair Advantage	Customer Segments
• It takes a lot of time to gather parking data from different sources in order to plan improvements in the structure (Traffic/City planners) • I don't have proper measurements of the impact of parking violation on the traffic flow (Traffic/City planners) • It's hard to reinforce law for parking violators because we don't have enough agents.	• Display actioanable parking data in a unique platform. • It provides mesaurements on parking violation and it's impact on traffic flow. • It identifies the violators automatically.	Our solution simplifies data colelction and automates law enforcement on parking violation. Optimising public resources management.	Can't be easily copied or bought	• City Planners • Traffic agents

Existing Alternatives	Key Metrics	High-Level Concept	Early Adopters
• Parking data is generated manually (by manual counting methods) • Data is brought from third parties like here maps • Fines are issues manually	• No of parking violations • Street where more number of parking violations happens	Our solution = Traffic flow optimisation And making parking space available for delivery vans	• Municipality of Hamburg

Figure 2.3: Lean Canvas
[source Roadlytics]

Figure 2.4: Roadlytics in a Nutshell
[source Roadlytics]

Chapter 3

Understanding Geospatial data

3.1 GPS and GNSS data

3.1.1 Working of GPS

Global Positioning System (GPS) is based on a group of 24 satellites and their ground stations developed by U.S. Department of Defence, every satellite sends data to its receiver in a form of signal which then computes it's position and time. There are at least 24 active satellites orbiting over 12,000 miles above earth on any given time, the signal information is transmitted over radio frequency ranging from 1.1 to 1.5 GHz. According to D. DOBERSTEIN [11] a GPS receiver typically needs to receive data from at least 4 satellites to determine it's position a very important note is that GPS receiver will not transmit any data to the satellites.

The GPS receiver receives a signal from each GPS satellite. The satellites relay the exact time the signals are sent out. By subtracting the time the signal was sent from the time it was received, the GPS could tell how far it was from each satellite. So given the travel time of the GPS signals from three satellites and their exact position in the sky, the GPS receiver can determine your position in three dimensions – east, north and altitude.

Imagine you are standing somewhere on Earth with three satellites in the sky above you. If you know how far away you are from satellite 1, then you know you must be located somewhere on the blue circle. If you do the same for satellites 2 and 3, you can work out your location by seeing where the three circles intersect. This is just what your GPS receiver does, although it uses overlapping spheres rather than circles. To calculate the time the GPS signals took to arrive, the GPS receiver needs to know the time very accurately. The GPS satellites have atomic clocks that keep very precise time, but it's not feasible to equip a GPS receiver with an atomic clock. However, if the GPS receiver uses the signal from a fourth satellite it can solve an equation that lets it determine the exact time, without needing an atomic clock. If the GPS receiver is only able to get signals from 3 satellites, you can still get your position, but it will be less accurate.

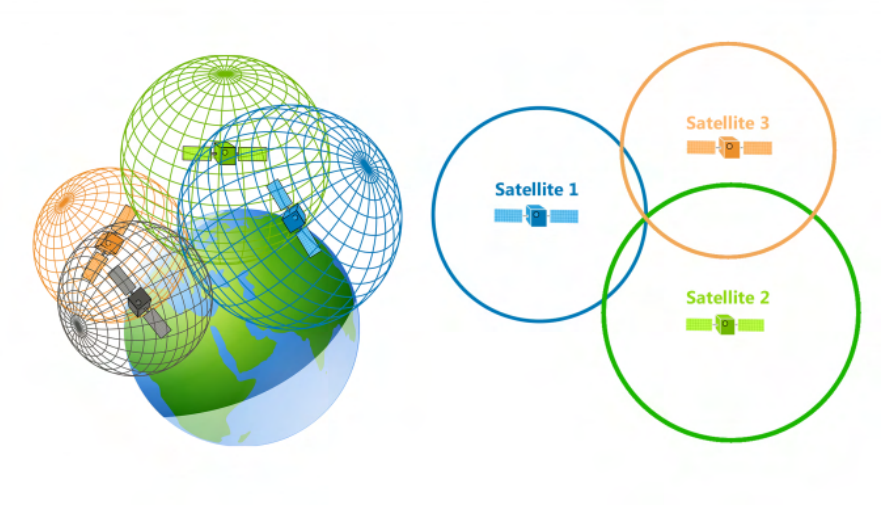


Figure 3.1: How GPS determines actual location of gps-receivers
[reproduced from GIS Geography]

Modern receivers like Global Navigational Satellite Systems (GNSS) have the capability to receive concurrent signals from more than 4 satellites at a given time without any trouble hence, making the position data accurate, getting a lock on by GNSS receivers on the ground takes time, particularly if the receiver is in a moving vehicle or in dense urban areas. The initial time required for a GNSS lock is normally determined by how the GNSS receiver boots up. There are three different types of starts: hot, warm, and cold they are all classified as "Acquisition Time". Acquisition Time is the time it takes a GPS receiver to lock onto enough satellites to have a position fix (three satellites for a 2D fix and four satellites for a 3D fix).

- The "hot start" is when the GNSS device recalls its last calculated location and the satellites in view, the almanac used (information about all the satellites in the constellation), the UTC Time, and attempts to lock onto the same satellites and calculate a new position based on the previous information. This is the fastest GNSS lock, but it only works if the device is in the same position as when the GNSS was last switched off.
- The "warm start" occurs when the GNSS system recalls its last measured location, almanac used, and UTC Time, but not which satellites were in view. It then resets, tries to acquire satellite signals, and calculates a new location.
- The "cold start" is the process by which the GNSS system dumps all of its information, attempts to locate satellites, and then calculates a GNSS lock. This is the most time-consuming because there is no prior information.

In the following section we will compare why GNSS is better than GPS receivers.

3.1.2 Difference between GPS and GNSS

GNSS and GPS work together, but the main difference is that GNSS-compatible trackers can use navigational satellites from other networks beyond the GPS system like Russian GLONASS, Chinese BeiDou, European Galileo etc more satellites means increased receiver accuracy and reliability. The accuracy of GNSS is far better than GPS receiver because at any given point of time GNSS receives signals from at least three operational satellite systems whereas GPS can receive signals form only one operational satellite system that is Global Positional Satellite. The primary difference between GNSS and GPS is, GPS uses 31 satellites within it's system where as GNSS uses 89 satellites within it's system, therefore GNSS receiver gets it's fix Time To First Fix (TTFF) within 60 seconds when its first powered on.

We collected sample data from GPS and GNSS receivers, plotted them on map for comparing the accuracy, Figure 3.2 show data from GPS receiver and Figure 3.3 display data from GNSS receiver, visually comparing the two figures, we can see data discrepancy in left side picture is more compare to right side because GNSS is more accurate in comparison to GPS receivers even in the harsh environments like weather conditions, tall buildings or urban congested areas.

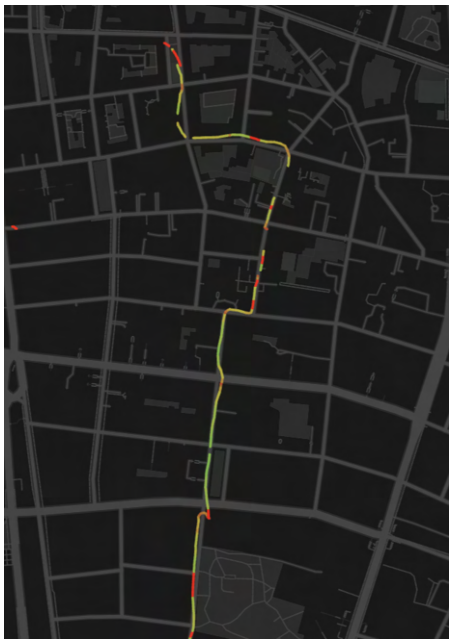


Figure 3.2: GPS-track

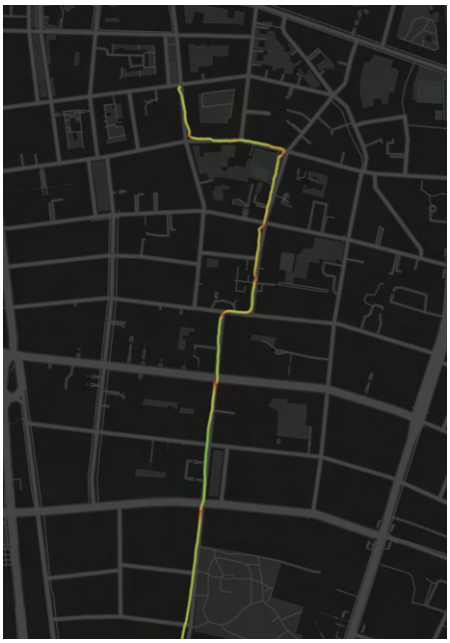


Figure 3.3: GNSS-track

Figure 3.4: Tracker data comparison
[source Roadlytics]

3.2 Georefernced data from GNSS receivers in the form of raw data

By default ublox-GNSS receiver generates NMEA sentences based on NMEA 0183 version standard as raw gnss data, NMEA stands for ***National Marine Electronics Association*** [12], the messages are in specific formats as shown in the figure 3.6, parsing these stream of sentences will shell out information like ***latitude, longitude, speed, altitude, timestamp,...***

3.2.1 Decoding single NMEA sentence

Figure 3.5 shows the NMEA sentences with variety of messages, each message contains certain details like

- $GNGSA - GNSS DOP and active satellite
- $GNGLL - Geographic Position, Latitude/Longitude
- **$GNGGA** - Global Positioning System Fix Data
- **$GNRMC** - Recommended minimum specific GPS/Transit data

Lets decode one of the following sentence, the decoded result are shown below table 3.2.1

```
$GNVTG,,T,,M,0.021,N,0.040,K,A*3A
$GNGGA,114302.00,5332.65600,N,01001.44657,E,1,05,2.96,46.1,M,45.1,M,,*7A
$GNGSA,A,3,30,,,,,,,,,,,,5.11,2.96,4.16*14
$GNGSA,A,3,73,70,87,72,,,,,,,,,5.11,2.96,4.16*1E
$GPGSV,2,1,05,02,,,26,05,,,18,16,01,018,,30,76,161,40*7A
$GPGSV,2,2,05,49,29,186,42*43
$GLGSV,2,1,07,69,01,124,,70,59,118,37,72,14,307,33,73,43,182,35*65
$GLGSV,2,2,07,86,03,261,,87,20,322,32,88,13,000,15*53
$GNGLL,5332.65600,N,01001.44657,E,114302.00,A,A*74
$GNRMC,114303.00,A,5332.65589,N,01001.44653,E,0.052,,151220,,,A*68
$GNVTG,,T,,M,0.052,N,0.096,K,A*35
$GNGGA,114303.00,5332.65589,N,01001.44653,E,1,05,2.96,46.1,M,45.1,M,,*7D
```

Figure 3.5: GNSS raw data

pynmea2 is a python library for parsing NMEA 0183 protocol, we implemented this parser to decode streams of incoming NMEA sentences to json format, the designed algorithm works in below fashion

225446	Time of fix 22:54:46 UTC
A	Navigation receiver warning A = OK, V = warning
4916.45,N	Latitude 49 deg. 16.45 min North
12311.12,W	Longitude 123 deg. 11.12 min West
000.5	Speed over ground, Knots
054.7	Course Made Good, True
191194	Date of fix 19 November 1994
020.3,E	Magnetic variation 20.3 deg East
*68	mandatory checksum

Table 3.1: Decoded '$GNRMC' sentence

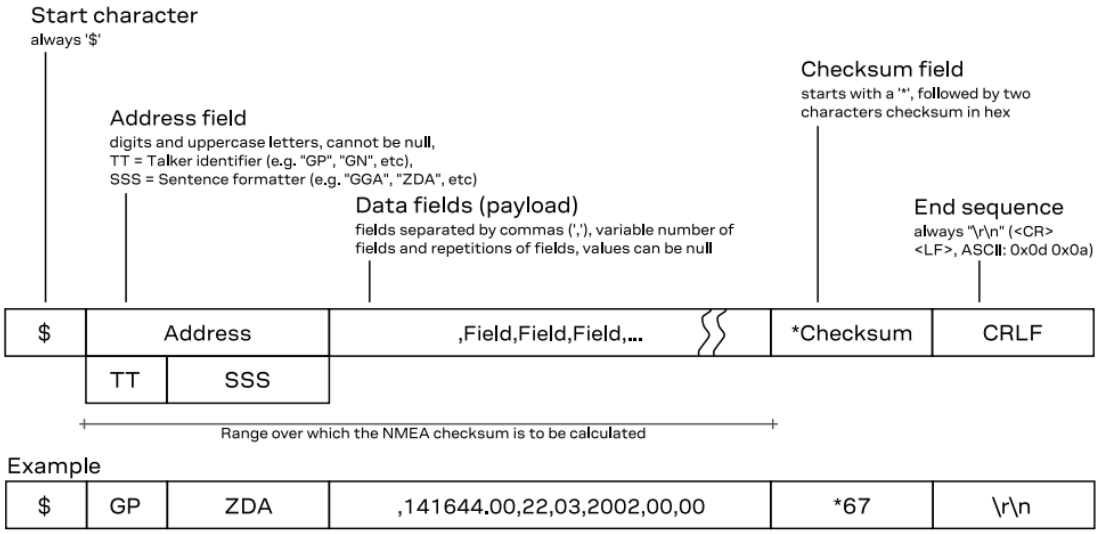


Figure 3.6: Structure of NMEA protocol message
[reproduced from u-blox.com]

- The algorithm waits until the receiver establishes strong connection with satellites (it should receive signals form at least 3 satellites to generate valid NMEA sentence).
- The stream of GNSS raw data (NMEA sentences) are passed to pynmea2 parser.
- The parser decodes the sentences into json file with following fields latitude, longitude, speed, date and timestamp.
- further analysis can be carried out with this data based on usecases.

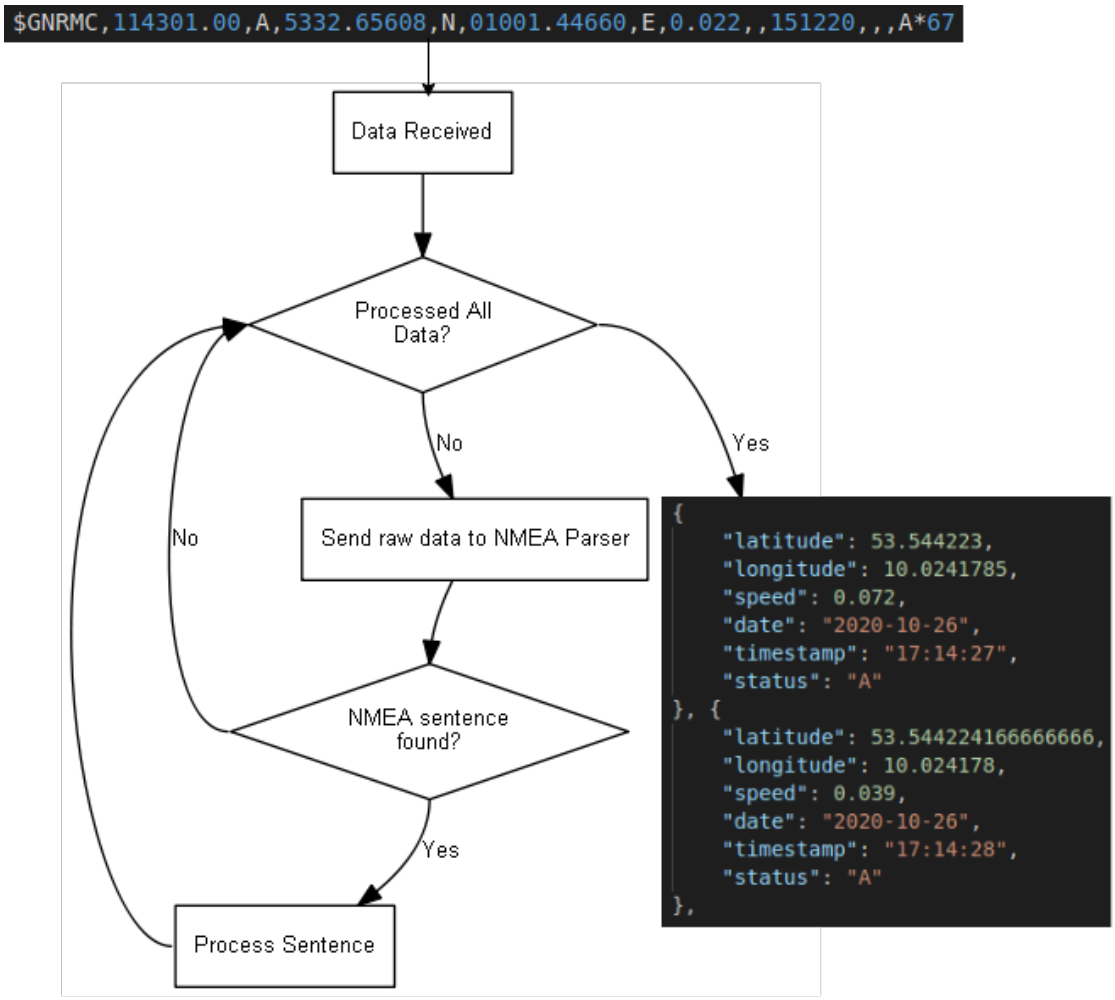


Figure 3.7: NMEA parser logic using python

3.3 Complete workflow of GNSS trackers in Roadlytics Application

Roadlytics project consist of two subsystems, firstly an IoT based GNSS tracker for data collection and secondly webapp dashboard for visualization and planning. The flowchart in the figure 3.8 shows wok flow of the system, As soon as the tracker gets power supply to be precise when the tracker is in ON state following things happens

- GNSS receiver established connections with multiple satellites to locate its position.
- The data received in tracker are currently in RAW NMEA format.
- The tracker waits until it gets complete set of NMEA sentence and save it as

log files.

- Log files are parsed using python's pynmea parser and converts the data from ASCII/HEX to JSON format.
- JSON files are transmitted to roadlytics backend server (MongoDB) using LTE-M and MQTT transmission medium.
- Roadlytics dashboard serves users requests based on their actions likes Route heatmap, Show hotspots etc

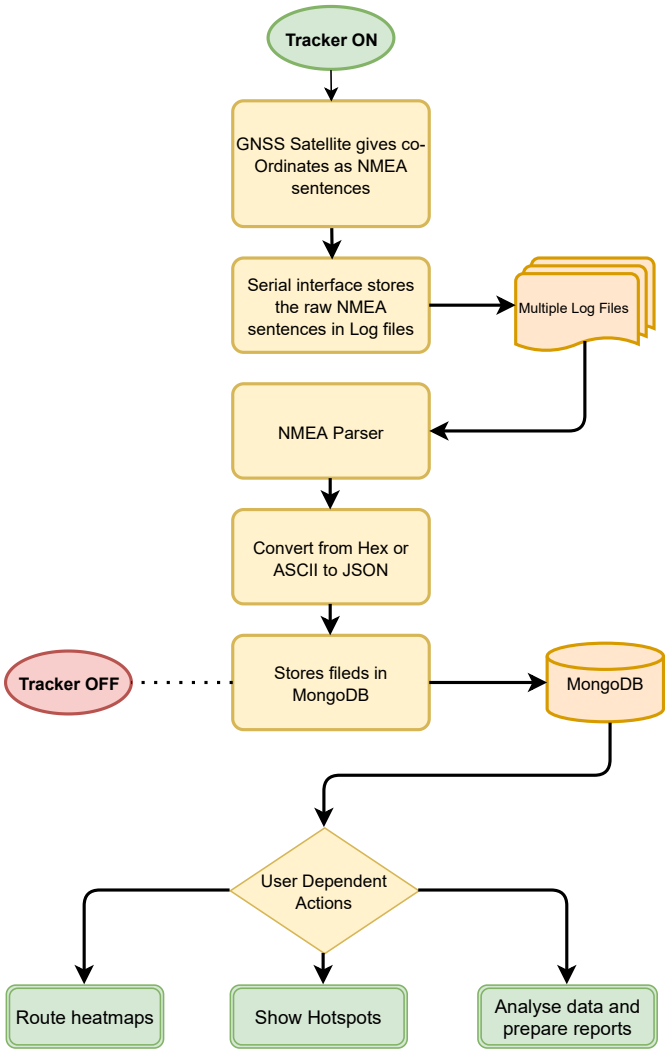


Figure 3.8: GNSS tracker workflow in Roadlytics Application [source Roadlytics]

Chapter 4

Hardware Prototypes

4.1 Prototype Version 1 - GNSS, Raspberry Pi 4 [GNSS Data Logger]

In this section, we will discuss how we built our first basic prototype with limited functionality. The following subsection will explain in detail each and every Hardware component used in building the prototype as well as the software stack used. The basic functionality of this prototype is to "log geo-referenced or position data" locally on to SD card whenever the device is turned on. The data is then processed for analysis and visualization. Let's take a deep dive into each subsection for a more thorough setup.

4.1.1 Hardware Requirements List

all the required hardware components are explained in this section

1. Raspberry PI 4

   Although there are many Development Boards available in the market at various prices, the Raspberry Pi is supposedly the best suited for IOT rapid prototyping. It's a credit card sized device, not a micro controller or microprocessor, it's a full computer with raspi os running on it [13], as seen in the figure 4.1 has many interfaces but let's look at the ones that are being used.

   - USB C Port [power supply]: For providing power supply to Pi, the power supply must be 5.1V or 3.0A DC output.
   - GPIO: This peripheral, which stands for General Purpose Input Output Pins, allows us to easily add extensions to the Pi by using male-female or female-female jumper pins.

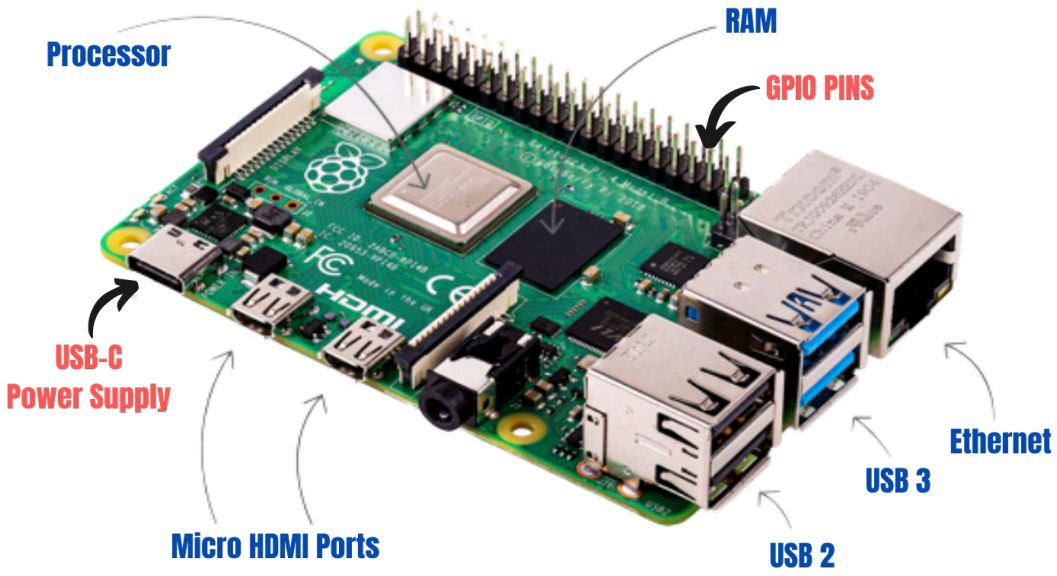


Figure 4.1: Raspberry Pi 4
[Adapted from: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/]

2. Ublox SAM-M8Q GNSS Receiver:

The GNSS 4 click comes with integrated Ublox SAM M8Q module with patched antenna, it runs with 3.3 V power supply and communicates with the raspberry pi over serial UART interface. The SAM-M8Q module supports concurrent reception of up to three GNSS systems like GPS, Galileo and GLONASS, recognizes multiple constellations at the same time, and offers exceptional positioning accuracy in situations involving urban or poor signals. The patch antenna is unaffected by its environment and has a high tolerance for frequency shifts. The u-blox M8 technology has a power-optimized architecture with built-in autonomous power saving functions to reduce power consumption at all times [14].

Figure 4.2: GNSS 4 click with integrated SAM-M8Q
[reproduced from: https://www.mikroe.com/gnss-4-click]

- PATCH ANTENNA

  The GNSS patch antenna is RHCP (right hand circular polarization) and has a peak gain of 3dB. The patch antenna is insensitive to surroundings and has high tolerance against frequency shifts.

- POWER MANAGEMENT

  u-blox M8 technology offers a power-optimized architecture with built-in autonomous power saving functions to minimize power consumption at any given time. Furthermore, the receiver can be used in two operating modes: Continuous mode for best performance or Power Save Mode for optimized power consumption.

- GOOD IN HOSTILE ENVIRONMENTS

  Thanks to all these features the SAM-M8Q module is good in GNSS-hostile environments - indoor spaces, urban canyons (when a street is flanked by buildings on both sides), etc.

- HOW IT WORKS

  A constellation of satellites sends a continuous signal towards earth, every satellite has an atomic clock, and all of them are synchronized, thanks to a reference time scale defined by the whole system. So, that the signals coming from the different satellites of the same constellation share the same reference time scale. The patched antenna helps to get the stronger signals and the receiver component of the module will translate these signals into NMEA sentences, which stands for GNSS

  If the user wants to utilize GNSS to determine a position, they must have an antenna that receives the signals coming from the satellites, and a receiver that translates these signals. The antenna position will be deduced from

the measurements of the time delay between the emission time (satellite) and the reception time (receiver) for at least 4 signals coming from different satellites.

- PIN OUT DIAGRAM
  Below figure 4.3 shows the pin out diagram of the GNSS module, only three pins are used as highlighted, pin-14 to transfer the GNSS NMEA sentences to the Pi, pin-7 for power supply, and pin-8 for ground.
  - TXD : to transmit UART data to raspberry pi
  - GND : for grounding
  - VCC: Power supply of 3.3v

Notes	Pin	mikroBUS				Pin	Notes
	NC	1	AN	PWM	16	NC	
Active Low	**RST_N**	2	RST	INT	15	NC	
	NC	3	CS	TX	14	**TXD**	UART transmit
	NC	4	SCK	RX	13	**RXD**	UART receive
	NC	5	MISO	SCL	12	**SCL**	I2C Clock
	NC	6	MOSI	SDA	11	**SDA**	I2C Data
Power supply	**+3.3V**	7	3.3V	5V	10	NC	
Ground	**GND**	8	GND	GND	9	**GND**	Ground

Figure 4.3: Pinout Diagram
[adapted from: https://www.mikroe.com/gnss-4-click]

3. Powerbank:

A 20000 mAh Lithium Polymer Ion power bank with 5v 2Amps output is used for the portable power supply to the entire device.

Figure 4.4: From Left: LiPo Powerbank, USB C Type Cable, SD Card, Jumper Wires [reproduced from: https://www.conrad.com/]

4. Jumper Wires female to female: These wires will help us to connect Rx and Tx port of the GNSS module with the GPIO pins of the raspberry pi.
5. Micro USB cable: This will connect between the power port of the raspberry Pi Zero wireless and the power bank.
6. Sandisk SD Card: The OS is booted on SD card and also serves as a storage drive with core logic running on it.

4.1.2 Hardware Configuration and setup

Flashing Raspberry Pi OS on SD card, enabling SSH for easy logins, Network configuration on boot and enabling serial port for data transmission and receiving (Tx/Rx)

Raspberry Pi OS is a free operating system built on Debian that was designed specifically for the Raspberry Pi hardware. We are using "Raspberry Pi Imager" a graphical SD card writing tool developed by pi community [15] to burn the OS on SD card. The steps are as follows

- Download the latest version of Raspberry Pi Imager utility and install it on a computer with linux or windows. Type the command to install it on linux

```
$ sudo apt install rpi-imager
```

- Insert SD card reader into the computer with the SD card inside.
- Open Raspberry Pi Imager and choose the required OS from the list. Below command for linux

```
$ rpi-imager
```

- Choose the SD card you wish to write your image to.
- Review your selections and click on 'write' button, the process begins and upon completion a success message will pop up.

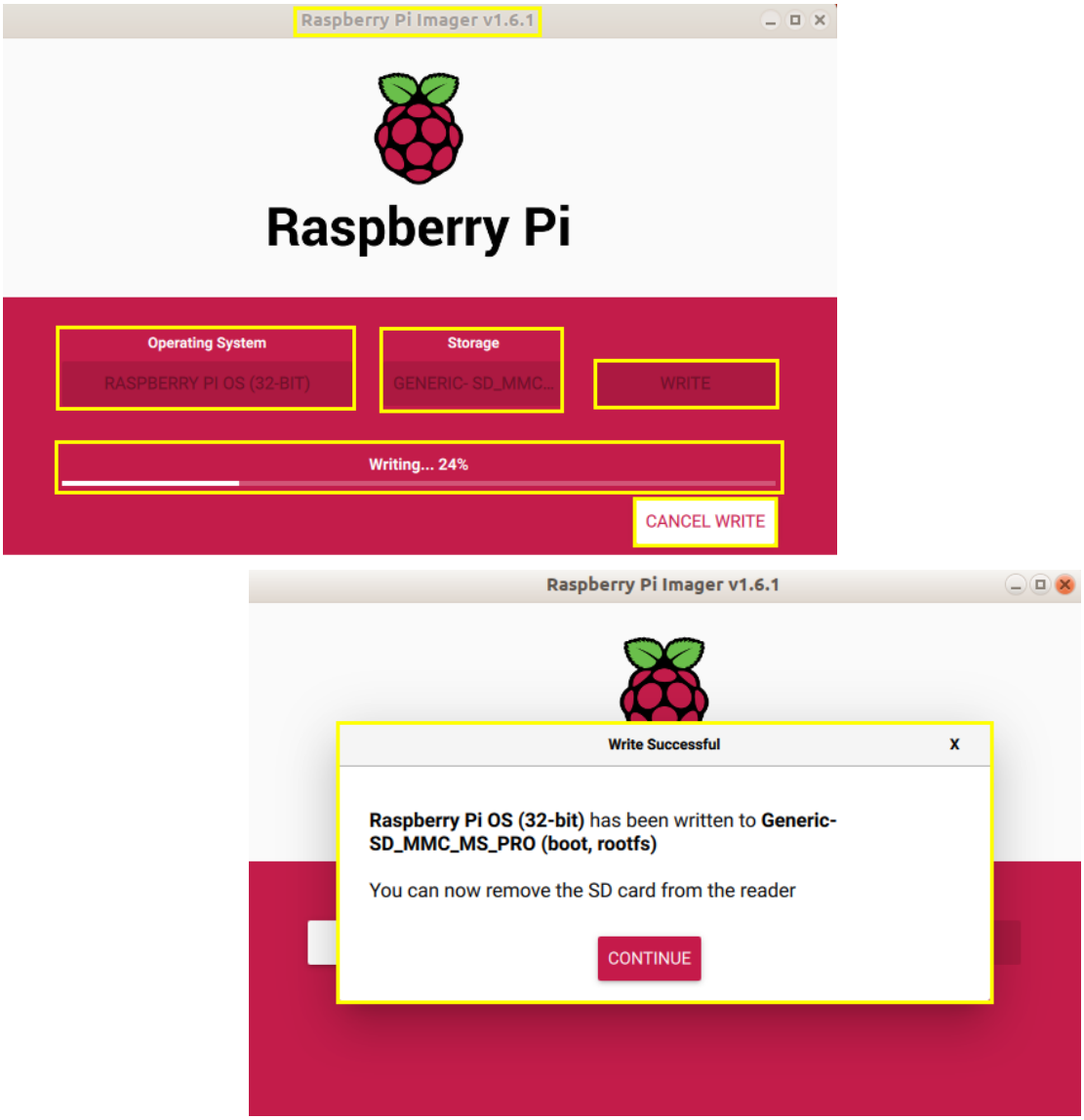
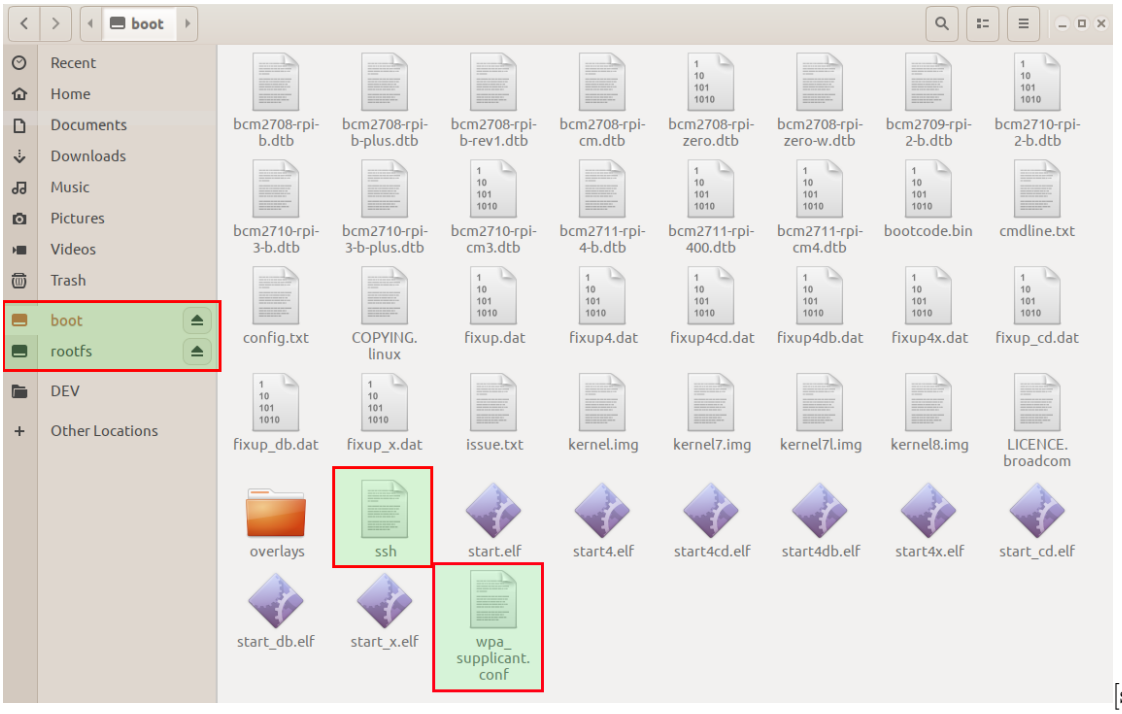


Figure 4.5: Writing Raspberry Pi OS on SD card using rpi-imager utility [source Roadlytics]

- Upon completion, the SD card will have two partition in it as shown in the figure 4.6 "boot" and "roofts", to enable SSH in case of 'headless pi', create an empty file with name 'SSH' without any extension and to automatically connect raspberry pi to internet on boot, create another configuration file with name 'wpa_supplicant.conf' and include the network details as shown below

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=DE
```

```
network={
    ssid="<YOUR NETWORK NAME>"
    psk="<YOUR NETWORK PASSWORD>"
    key_mgmt=WPA-PSK
}
```



[source Roadlytics]

Figure 4.6: Enabling SSH and network on boot

- Once the above steps are complete, insert the SD card back into raspberry pi's dedicated slot and power up the device, the first time boot will take approximately 5 mins, on successful boot up the power led light stops blinking
- Now, assuming the pi has already connected to the local wifi network, we can SSH pi using the default credentials

  ```
  ssh pi@raspberrypi
  ```

- Pi and GNSS receiver communicates via serial port, hence enabling it to receive stream of georeferenced data as shown in the below figure 4.7 and 4.8

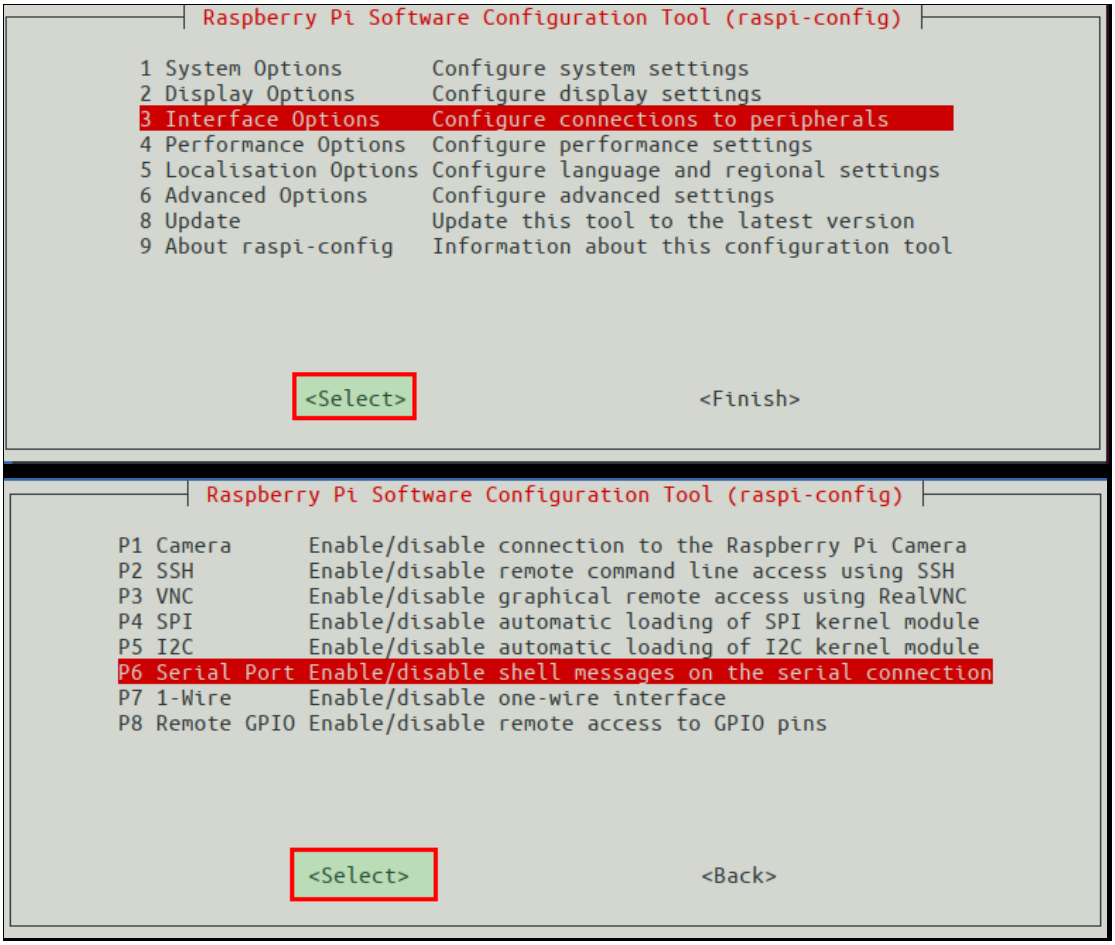
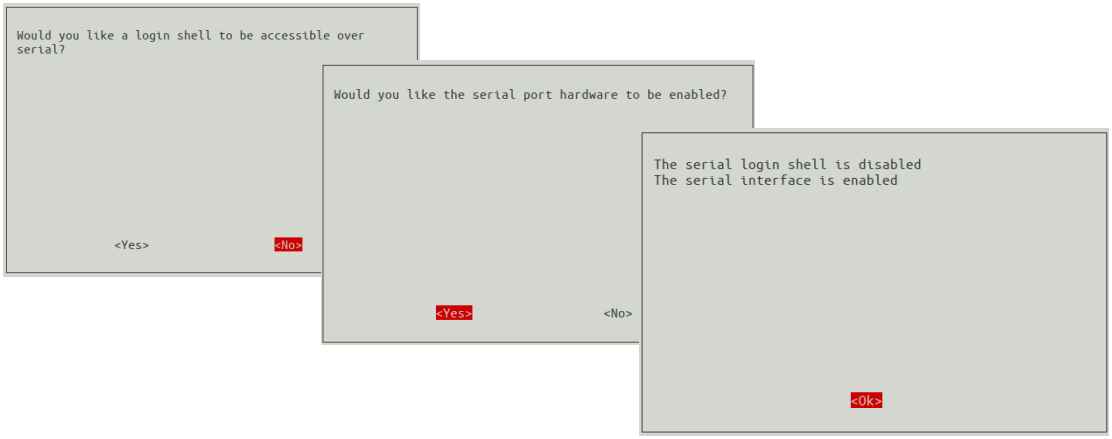


Figure 4.7: Enabling Serial Port
[source Roadlytics]



[source Roadlytics]

Figure 4.8: Enabling Serial Port

4.1.3 Assembling Hardware Modules and Setting up Software

Assembling Hardware Modules

In this section, we will combine all of the previously mentioned Hardware Modules into one complete package. The next step will be to install, configure, and test the device to ensure that it works as intended. The final design of Prototype Version 1 is shown in the figure 4.9. The Raspberry Pi and GNSS receiver are connected using female to female jumper pins as shown in the Pin link table 4.1, and a power supply is provided using a power bank.

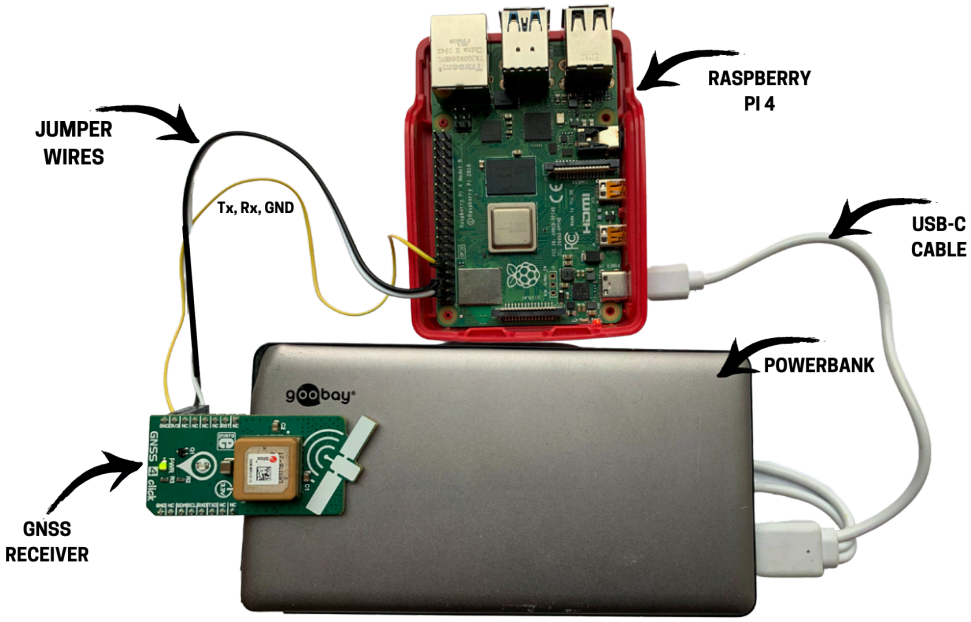


Figure 4.9: Prototype Version 1
[source Roadlytics]

	GPIO Pins	GNSS Receiver
Data Transfer	Rx	Tx
Power Supply	VCC	3V
Ground	GND	GND

Table 4.1: PIN connection

Testing Serial Port Communication

We have already enabled Serial Port in the above installation process, now we will test if the serial port is able to Receive raw NMEA sentences from GNSS receiver, as shown in the figure 4.10 just by typing the following command we can see the stream of data in the terminal, which confirms the serial port is open and device installation is perfect.

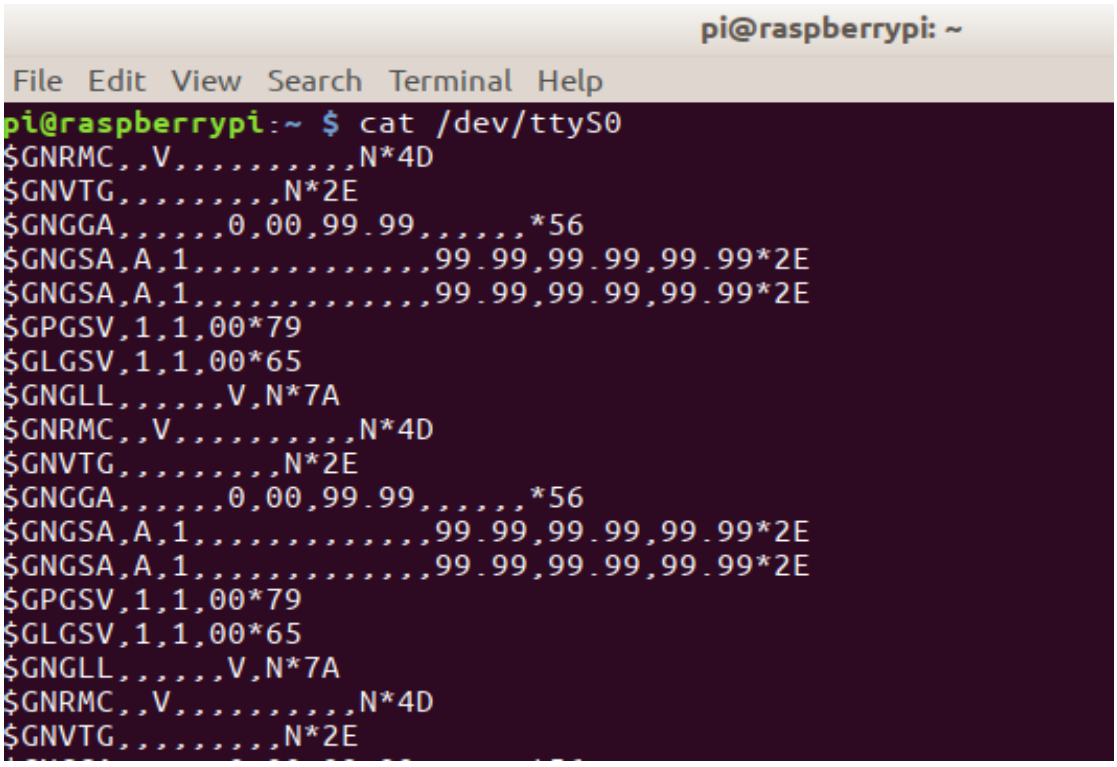


Figure 4.10: Data generated form Serial Port
[source Roadlytics]

Automating to run the python script on boot

Here we are using CRONTAB utility, the crontab is a list of commands that are scheduled to run at the given regular intervals, The crontab command opens the crontab-editor to add, remove or modify scheduled tasks, the daemon which reads the crontab and executes the commands at the right time is called cron.

```
# To open a crontab, just type the command

$ sudo crontab -e

# A file will be opened, add the below line of code at
the end of the file to start the script on boot

@reboot sudo python /pathToSourceCode/GNSS_tracker.py
```

Working Prototype Version 1 and it's Software Stack

When the device is switched ON for the very first time, pi takes approximately 30 seconds to bootup and in the meantime GNSS receiver take 26 seconds for

"COLD START", this GNSS term "cold start" TTFF, for example, is the time it takes to achieve a position fix after a complete power-off with no memory, while a "HOT START" TTFF is the time it takes to achieve a position fix after a short period power-off with internal memory and timekeeping to maintain critical data and time information.

When the device is up and running, the background script spins up and parses the NMEA sentences which are transmitted from ublox-sam-m8q receiver via UART protocol and the data is stored in the local file system. For every device bootup a new data file is created to keep the process simple and the data file is labelled with the startup timestamp hence, whenever a user turns off the device and turns on again a new data file will be created and stored in the Pi's file system, later these files are extracted through SSH, processed and visualized.

A simple architecture or software stack is shown in the figure 4.11, at the bottom of the software stack is the U-blox SAM M8q GNSS receiver module that communicates through the Pi's General Purpose Input/Output (GPIO) pins to the software that is running on the Pi. At the top of the software stack is the application code purely written in python, which makes calls into the middle stack where python NMEA parser parses raw NMEA sentences to JSON format and finally stores into the local file system. The complete source code is given in the appendix 7.1.

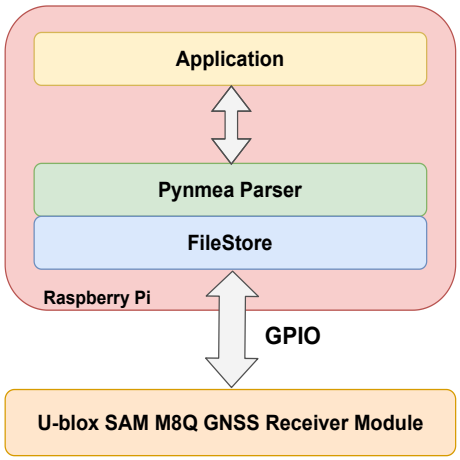


Figure 4.11: Architecture of Prototype 1
[source Roadlytics]

4.1.4 Data Processing and Visualization

on ubuntu SSH is accessed using terminal program similarly on other operating system "PuTTY" can be used, to login to Pi, ssh requires pi's username and password, once the access is granted we can copy the data files in to laptops local, the data pre-processing steps includes

- deleting faulty data

- formatting json structure [the tool used is called jsonlint[16]]
- upload the json to kepler's GUI[17]

** Include Pi's internal folder structure and few sample data with kepler map **

4.2 Prototype Version 2 - GNSS module, Raspberry Pi ZERO Wireless [Upgraded GNSS Data Logger]

Compare to the first Version, in this prototype 2 we have replaced raspberry pi 4 board with raspberry pi zero wireless, as it has many benefits over raspberry pi 4, there are no changes in the application source code and circuit connections but the main reasons to choose raspberry pi zero wireless are as follows

- It is affordable: raspberry pi 4 cost approximately 40€ and raspberry pi zero costs 10€, by replacing the module there are performance changes but we end up lowering development cost.
- Raspberry pi zero is 40% faster compared to raspberry pi 4
- Raspberry pi zero comes without populated GPIO pins, hence giving the flexibility to use only the connections as per project requirement
- Consumes less space, raspberry pi zero measures 56mm long by 30mm wide and 5mm deep 4.13
- It consumes less power
- Quick boot time

4.2.1 Hardware Requirements List

The only change in the hardware is replacing raspberry pi 4 with raspberry pi zero wireless, figure 4.12 is pi zero with supporting functionalities labelled with different colors

4.2.2 Working Prototype Version 2 and it's Software Stack

This prototype works exactly the same way as previous version in spite of replacing the core hardware module, on the other hand it is much efficient then prototype version 1, it follows the same software stack as previous version, the final look and feel of the version 2 is shown in the picture 4.14

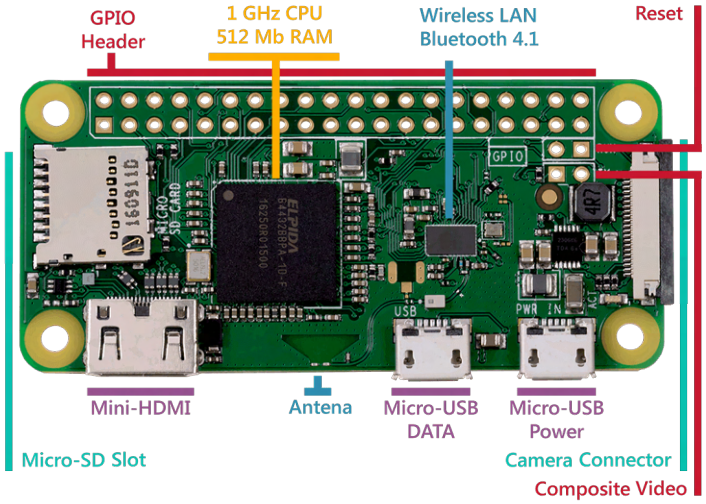


Figure 4.12: Raspberry PI ZERO W
[Adapted from: https://robu.in]

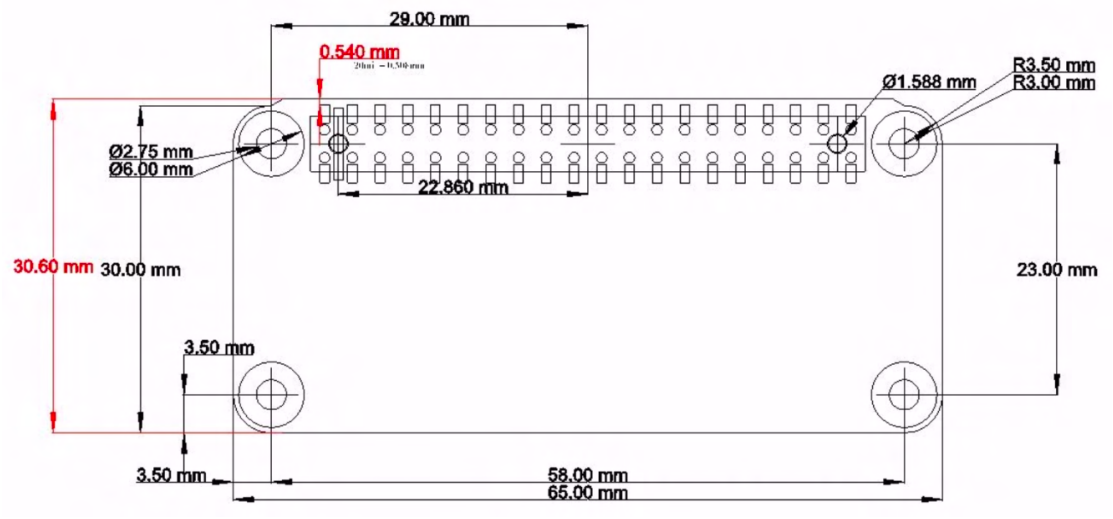
4.2.3 Data Processing and Visualization

4.3 Prototype Version 3 - GNSS, raspberry pi zero wireless and pi battery hat with initial state IOT platform

This is more sophisticated prototype version compared to previous two versions, there are two big modifications made in this prototype firstly, the power bank which weights approximately 430 grams have been replaced with 'Battery hat' which is less expensive and weights just 43 grams and has the same power output. Secondly, this device acts as real time GNSS/GPS tracker unlike GNSS/GPS logger which was our previous version, to achieve this functionality we use 'initial state'[18] IOT platform and a modified python script refer the appendix **??**.

4.3.1 Hardware Requirements List

There is no change in the raspberry pi zero wireless board and GNSS module (the details are already explained in prototype version 2), the only hardware change is the pi battery hat. This prototype is more compact and portal with the total weight of 67 grams, the only drawback of this version is the device works excellent for about 2 3 hours when fully charged.



[source Roadlytics]

Figure 4.13: Raspberry Pi zero measurement

Li-Ion Battery Hat for Raspberry Pi

The Li-ion Battery HAT incorporates the SW6106 power bank management chip, allowing the Pi to receive 5V controlled power from a 14500 battery, transforming the Pi into a portable unit. It also charges the battery and supports bi-directional fast charging. This module functions as a universal mini power bank and is compatible with other 5V products.The SW6016 is a highly integrated power management IC for fast charge power bank application, it integrates 4A switching charger, 18W synchronous boost, PD,QC,FCP,PE fast charge protocol, fuel gauge and power controller. It is shown in the figure 4.15 adapted from waveshare[19]

Interface

The battery hat got 3 interfaces and here 4.2 is the description of each interfaces and the highlighted one is used currently for this prototype version

Interface	Description
USB Type-C	Battery charge/Power output interface
USB Type-A	Power output interface
Micro USB	Battery charge interface

Table 4.2: Battery hat for Pi interface description

LED indication

The battery hat consist of 5 LED lights indicating the battery level when charging and discharging the module, it is clearly shown in the tables 4.4 and **??**

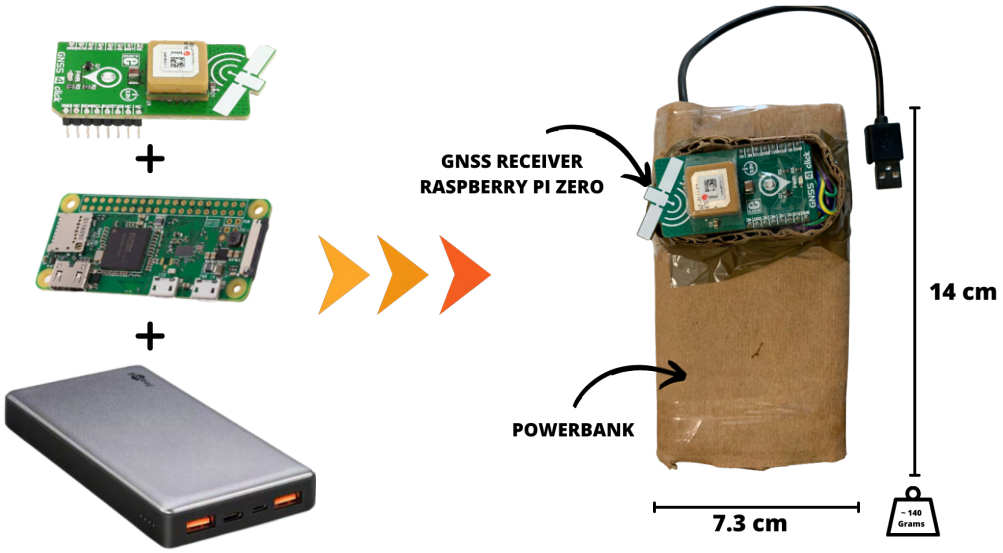


Figure 4.14: Prototype Version 2
[source Roadlytics]

Capacity	LED1	LED2	LED3	LED4	LED5
80~100%	ON	ON	ON	ON	ON
60~80%	ON	ON	ON	ON	OFF
40~60%	ON	ON	ON	OFF	OFF
20~40%	ON	ON	OFF	OFF	OFF
5~20%	ON	OFF	OFF	OFF	OFF
1~5%	Flicker	OFF	OFF	OFF	OFF
0%	OFF	OFF	OFF	OFF	OFF

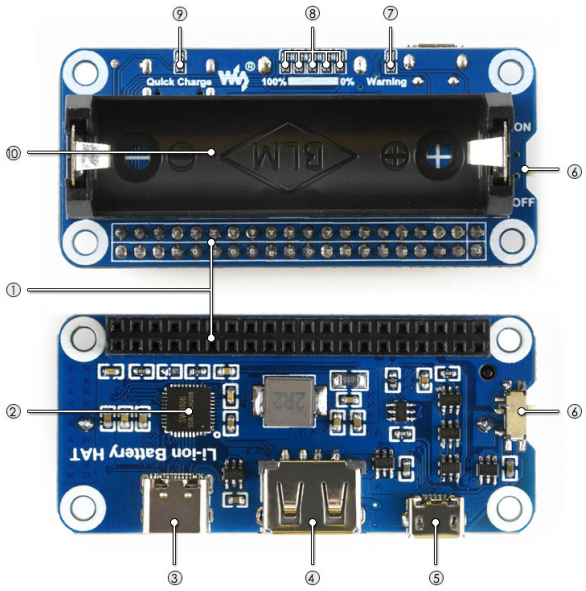
Table 4.3: LED states when discharging

4.3.2 IoT platform - Initial state

IOT platform is a software tool that connects "things" in an IoT ecosystem, "thing" hererefers to an IoT device, some of the basic features available in every IoT platforms are

- Device Management: It enables device provisioning, authentication, monitoring, diagnosing, device firmware updates, remote configuration, remote management last but not least device maintenance functionalities.
- IoT Application: These are nothing but streaming data from the things can be visualized on dashboards and these dashboard are customizable based on specific usecases of a project.

Initial State on the other hand is a cloud based IoT platform that accepts streaming data form any IoT device, these data can further be used to create an



1. **GPIO**
2. **Power Management Chip**
3. **USB Type-C**
4. **USB Type-A**
5. **Micro USB**
6. **Power Switch**
7. **Battery Warning Indicator**
8. **Battery Power Capacity Indicator**
9. **Quick Charging Indicator**
10. **14500 Battery Holder**

Figure 4.15: Li-Ion Battery Hat for Raspberry Pi
[Adapted from: Adapted from: waveshare.com]

Capacity	LED1	LED2	LED3	LED4	LED5
100%	ON	ON	ON	ON	ON
80~99%	ON	ON	ON	ON	Flicker
60~80%	ON	ON	ON	Flicker	OFF
40~60%	ON	ON	Flicker	OFF	OFF
20~40%	ON	Flicker	OFF	OFF	OFF
0~20%	Flicker	OFF	OFF	OFF	OFF

Table 4.4: LED states when discharging

interactive dashboards, we can also set triggers on critical thresholds, the processed data or the dashboard data can be shared between the platforms or databases and finally it also gives us an ability to monitor application in real time.

Initial State IoT platform and Data Visualization

Here we will discus how to set up an account on initial state and start receiving the data from the prototype and build interactive dashboard for analysis. The steps are

- Go to https://www.initialstate.com/ and register a new account shown in figure 4.16.
- Once you login, create a new "Stream Bucket" shown in figure 4.17.
- Type a relevant name ("GNSS") and click on create button, it will create fol-

lowing keys "Bucket key", "Access Key" and "API Endpoint" as shown in the figure 4.18, we will need to update these keys in the source code provided in the appendix section 7.2.

- once the above steps are complete, restart the raspberry pi and you will see the a dynamic map widget is created on initial state and the positions data can be visualized as shown in the figure 4.19.
- Now we can add custom tiles or edit the current tiles as per the requirements.

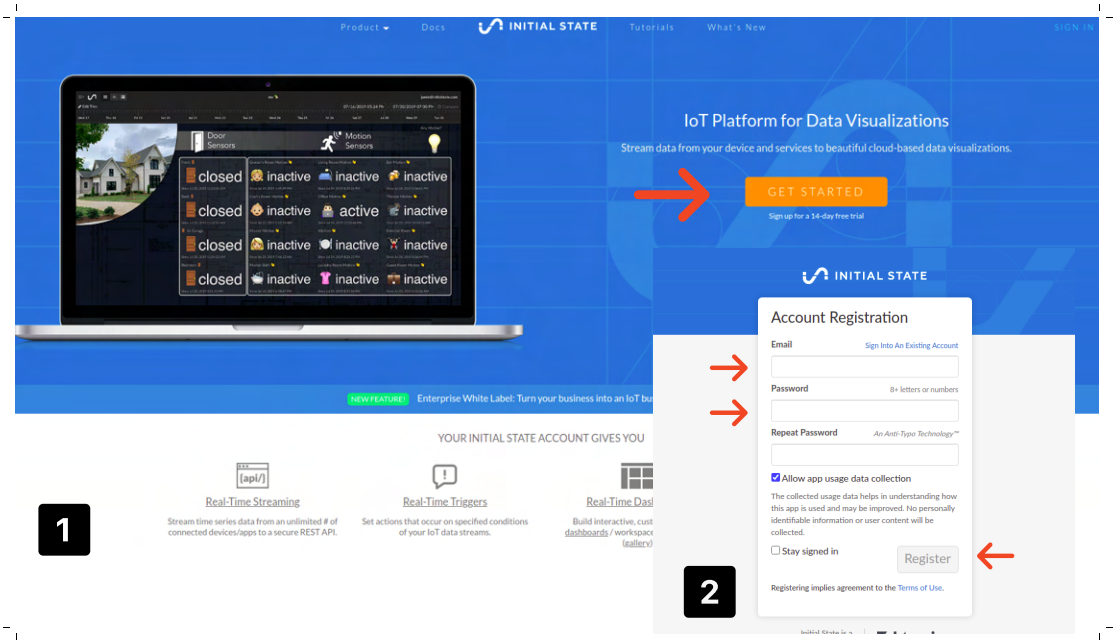


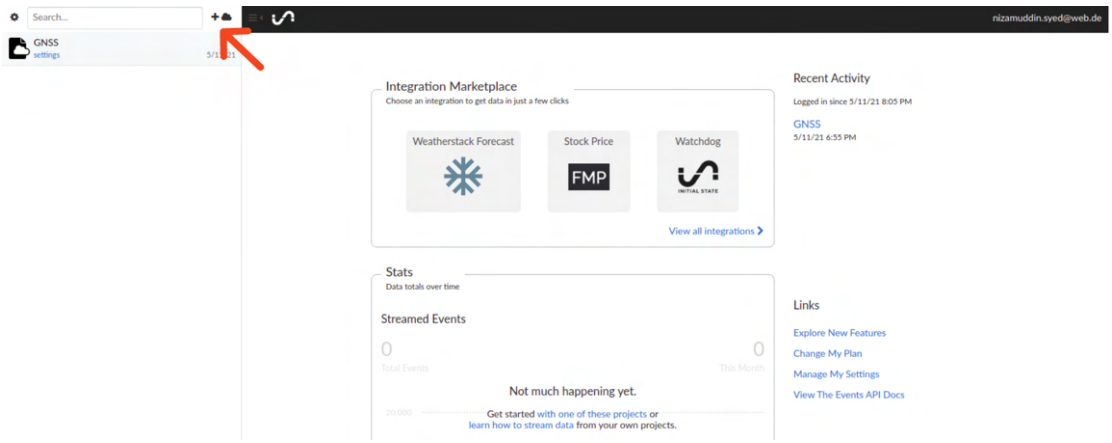
Figure 4.16: Initial State account creation and login

Prototype Version 3 and it's Software Stack

Finally, the the version 3 prototype is very lightweight, portable and a real-time position tracker, a user needs to just switch on the button and anyone having access to "initial state" can track the device, in our case we use mobile wifi hotspot connection to send the real time stream data to initial state server. The look and feel of the complete prototype is shown in the figure 4.20 and 4.21

4.4 Prototype Version 4 - RAK7200 raspberry pi 4 and RAK2245

The following prototype has a completely new hardware setup; the only module from previous prototypes that has been reused is the Raspberry Pi 4 that functions



[adapted form initial state]

Figure 4.17: Bucket Creation

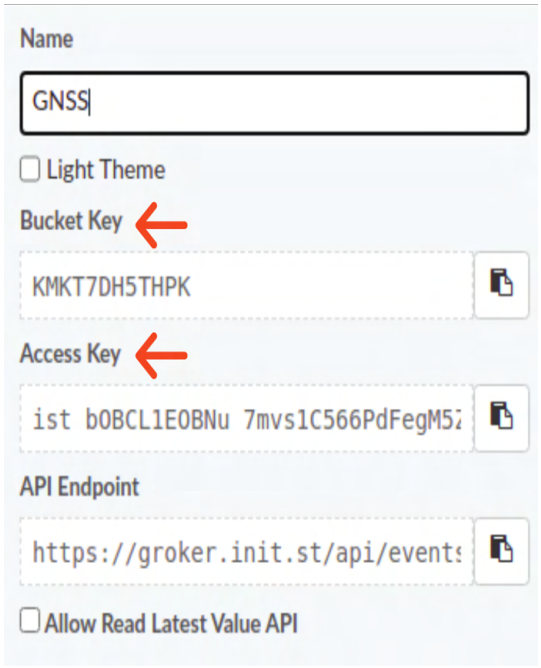


Figure 4.18: Initial State Key generation
[adapted form initial state]

as a LoRa-Gateway in conjunction with the RAK2245 Gateway module; here, the Long Range Network (LoRa) acts as a network communication channel to transfer data from nodes to servers and vice versa.

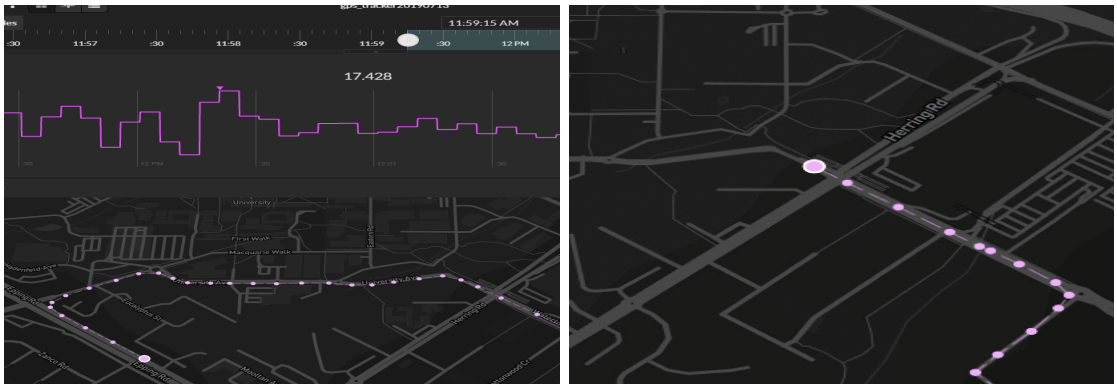


Figure 4.19: Initial State data streaming and visualization
[adapted form initial state]

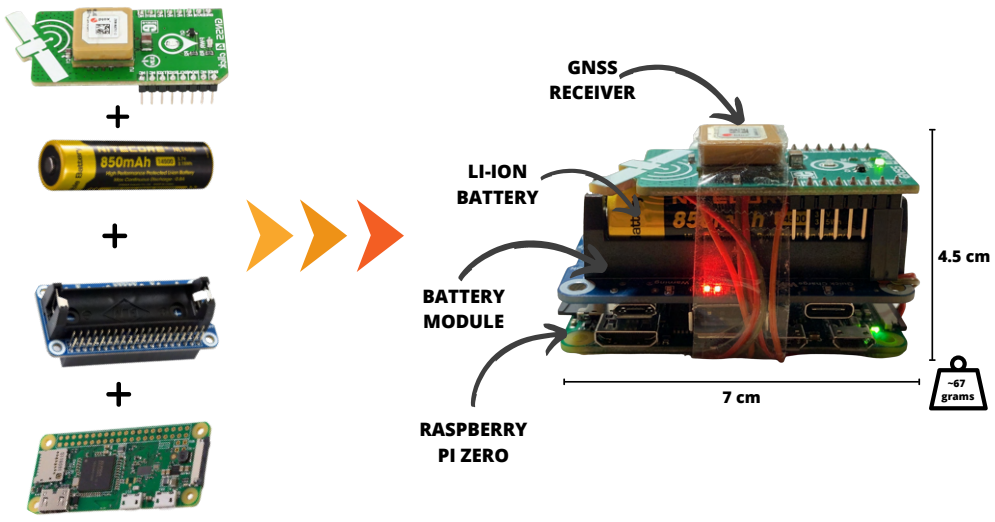
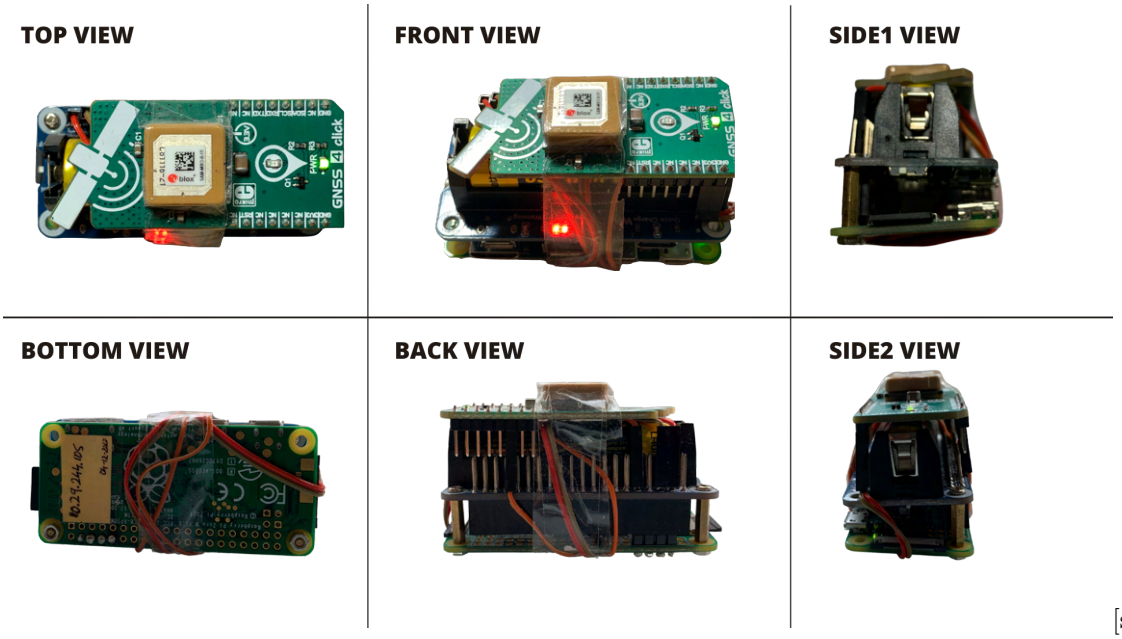


Figure 4.20: Prototype version 3 - realtime GNSS/GPS tracker
[source Roadlytics]

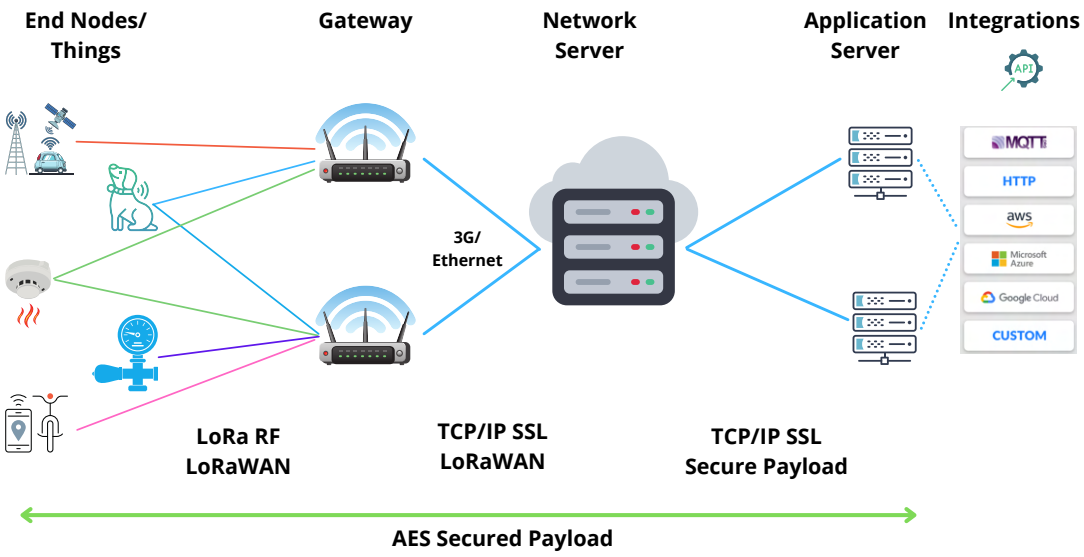
4.4.1 Hardware Requirements list

This prototype's hardware includes three main components: a "Node", which is an LPWAN based GPS tracker, a Raspberry Pi 4 and a RAK2245, which will sever as a LoRa-Gateway. Before we go any further, let's define the words **"LoRa" and "LoRaWAN."**



[source Roadlytics]

Figure 4.21: Prototype version 3 - dimension views



[source Roadlytics]

Figure 4.22: LoRa Network Architecture

LoRa is the physical layer or wireless modulation that is used to establish a long distance, low power communication connection [20]. It is normally built

around battery-powered radio units linked to sensors that can run for years without needing to be recharged.

LoRaWAN is a wireless protocol that is intended for use in the development of IoT networks. IOT devices (also known as "nodes" or "things") sends small data packets to a network of "Gateways" spread over many kilometers. The Gateways then route the messages to a Network Server, which validates the packets and routes the data payload to an Application Server using more traditional networking methods, such as wired Internet links.

The LoRa network architecture is depicted in the figure 4.22, in which LoRa compatible devices are referred to as nodes; these devices can be GPS tracking devices, pet tracking devices, smoke alarms, and many more; these devices are battery powered and can last for many years due to the LoRaWAN's powerful technology. The Node transmits data packets via radio frequency to the nearest public or private Gateway. There are already several thousand community Gateways available around the world that are open to the public, but private Gateways are more secure in terms of coverage and packet loss.

In the current prototype, we built our own Gateway using the RAK2245 Pi hat to route the packets received from nodes to the Network server(TTN), which validates the authenticity and integrity of the devices and packets. Ultimately, the application server is in charge of decrypting/encrypting data packets, and other IoT cloud systems such as AWS, Azure, and GCP can be integrated with the application server for further analysis.

RAK7200 - LoRa Node or End-device

The RAK7200 Node is an LPWAN compatible GPS tracking device with a 3.7V rechargeable Li-Ion battery and a GPS modem integrated into the casing. The payload is transmitted via Radio Frequency (RF) using the LoRaWAN protocol to a cloud-based TTN backend server. Each country has its own RF requirements, for example, EU686 in Germany. The RAK7200 module is depicted in the diagram 4.23.

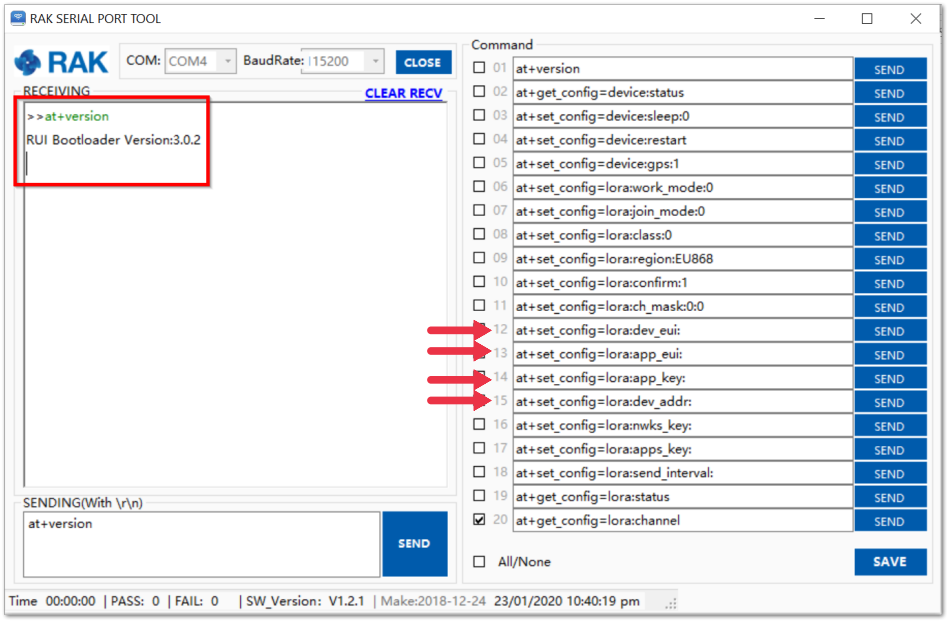
The device comes with pre-installed firmware, we must activate it by registering it on TTN console in order to establish the connectivity, as well as by adjusting the Node's configuration parameters using AT commands with the aid of a Windows application named "RAK SERIAL PORT TOOL," as shown in the screenshot below 4.24.

Connecting to the Things Network(TTN)

The registration of the device on the TTN database is an essential step for seamless communication; the steps to get the device up and running are outlined below.

1. Login to the TTN website and navigate to the console window; if no account exists, create one as shown in the picture 4.25.

Figure 4.23: RAK7200 Node
[adapted from RAKWireless]



[adapted from RAKWireless]

Figure 4.24: RAK7200 device setup and configuration

2. In image 2 of 4.25, the "Application tab" is nothing more than a node-application; select the Application tab and Add an Application as shown in the figure 4.26.

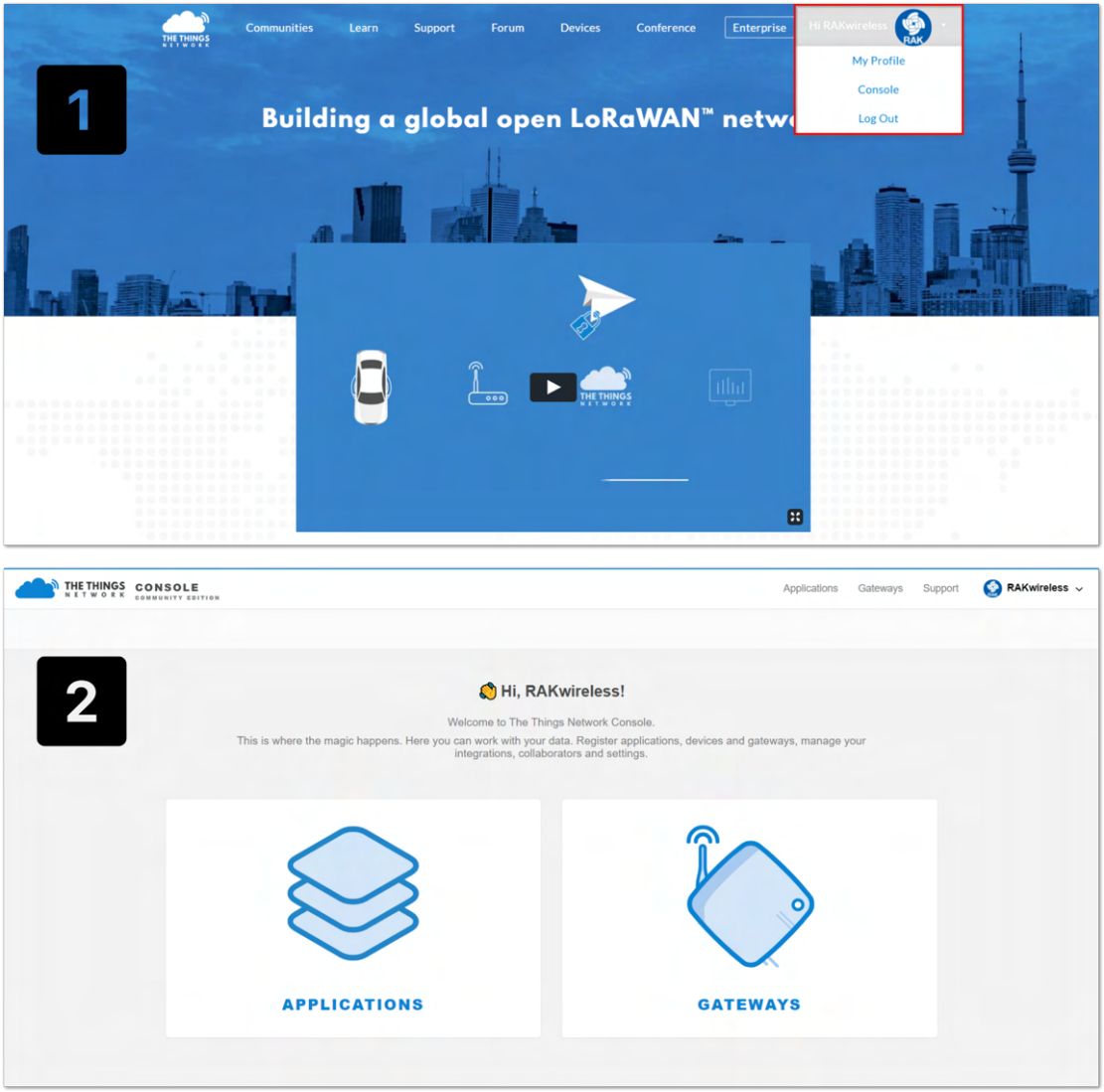


Figure 4.25: TTN Home Page and Console Page
[Adapted from www.rakwireless.com]

3. Here are some aspects to remember when adding an application.
   - **Application ID** - This will be the application's Network ID. Only lowercase letters and no spaces are permitted.
   - **Description** - A definition of an application.
   - **Application EUI** - TTN will produce this automatically.
4. After entering all of the above details, click the "Add application" button to register the application.
5. After the application has been created, it is time to register the device by

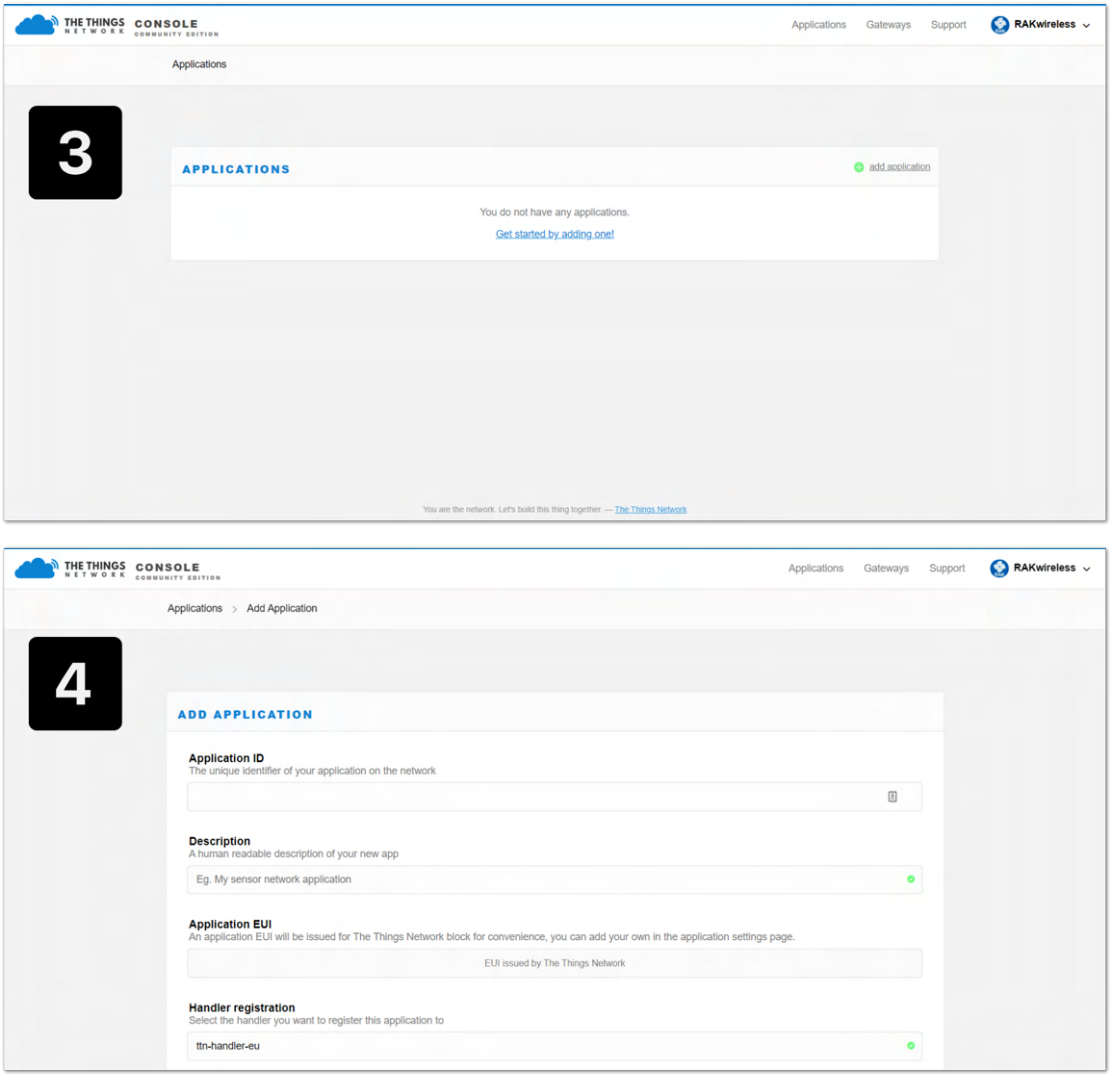


Figure 4.26: Adding an Application
[Adapted from www.rakwireless.com]

clicking the "register device" button and entering the correct Device ID. The rest of the parameters are generated at random, as shown in the diagram 4.27

6. Finally, as shown in the picture 4.28, press the "Register" button to complete the operation.

RAK2245 - LoRa Gateway

RAK2245 is a raspberry pi hat edition that is compatible with the pi's GPIO pin header, making it simple to deploy LoRaWAN solutions with the pi as the base.

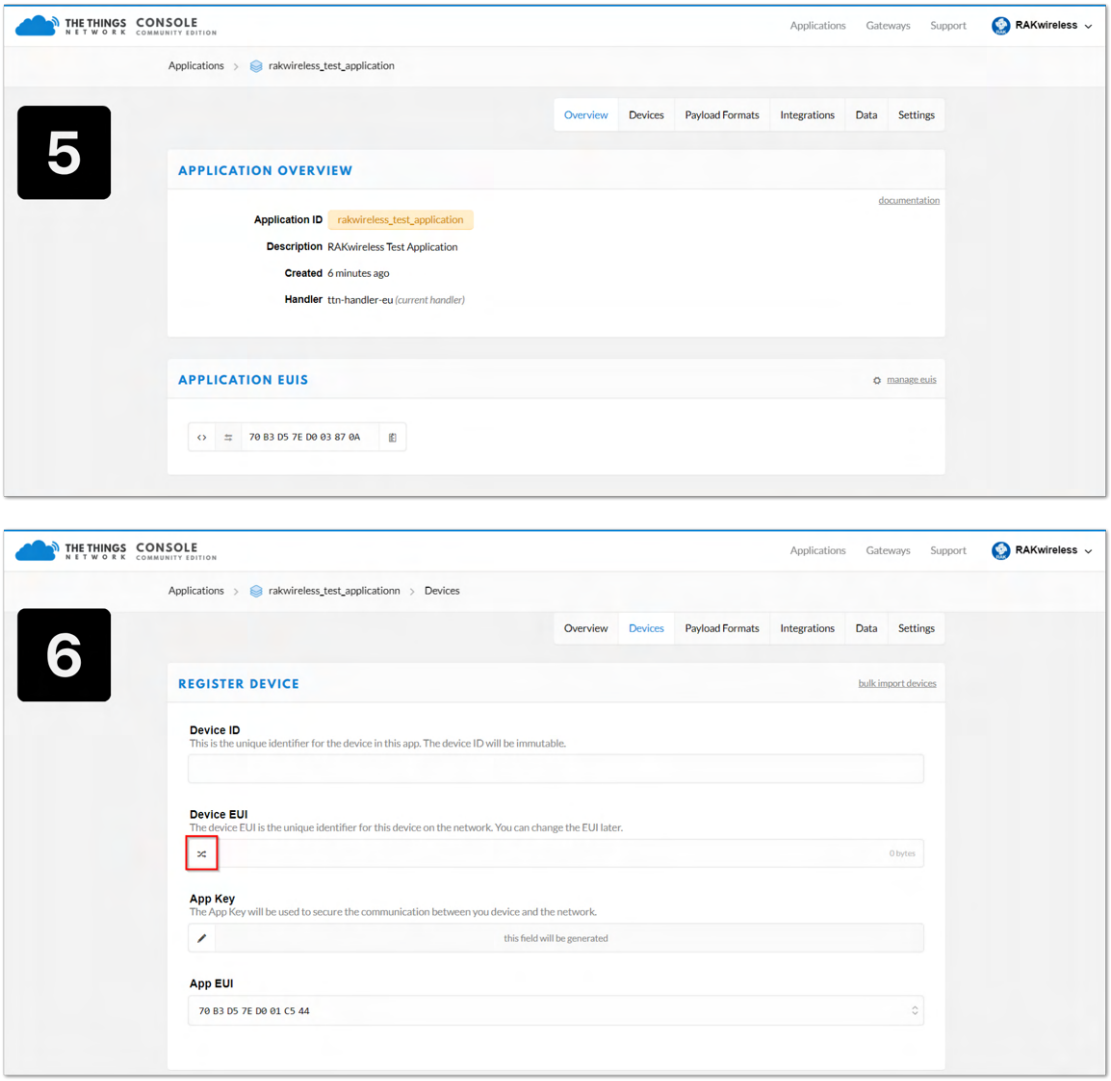


Figure 4.27: Registering and Adding Device
[Adapted from www.rakwireless.com]

It has eight channels and works with LoRaWAN's global frequency bands. The board is the smallest gateway concentrator on the market, and it includes the Ublox MAX-7Q GPS Module as well as a heat sink.

The board will have ultra-fast LoRa radio links with low data rates. It's operated by a Semtech SX1301 transceiver concentrator, which can handle packets from a large number of remote endpoints. Two Semtech SX125X RF front end I/Q transceivers are integrated.

The RAK2245 Pi HAT is a complete and cost-effective gateway solution for developing a complete LoRa system. This is a cost-effective way to develop smart grid, intelligent farm, and other Internet of Things applications.

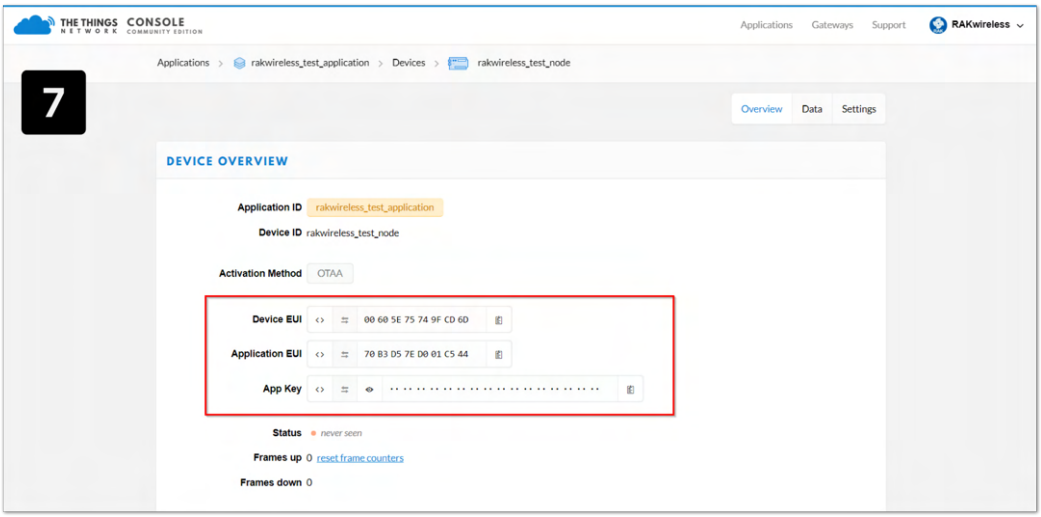


Figure 4.28: Device Overview
[Adapted from www.rakwireless.com]

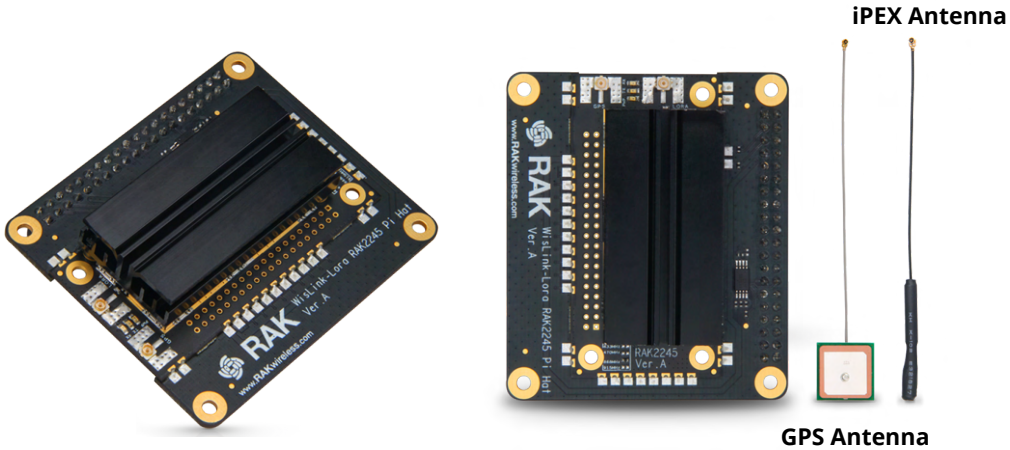


Figure 4.29: RAK2245 Pi Hat - LoRa Gateway Module

A Gateway acts as a link between nodes and TTN. Nodes use low power LoRaWAN network to connect to Gateway, while the Gateway connects to TTN via

high-bandwidth networks such as WiFi, Ethernet, or Cellular. All gateways within reach of a node can receive the Node's messages and forward them to TTN. The network will deduplicate the messages and choose the right gateway to forward any messages queued for downlink. A single gateway can support thousands of devices. The picture 4.29 depicts RAK2245 module

Gateway configuration and setup

The Gateway firmware can be downloaded from RAK's official website (`https://downloads.rakwireless.com/LoRa/RAK2245-Pi-HAT/`), then burn the SD Card with the new Firmware and plug it into the Raspberry Pi 4, then connect the RAK2245 and pi using GPIO pins, remember to connect the LoRa and GPS antennas. After that, we can turn on the Gateway safely. The Gateway will operate in Wi-Fi AP Mode by default, which means you'll see an SSID called "Rakwireless XXXX" in PC's Wi-Fi Network List. Connect to this Wi-Fi SSID by entering the password mentioned below.

```
Wi–Fi Password: rakwireless
Default IP Address: 192.168.230.1
Default Username: pi
Deafult Passsword: raspberry
```

Access the Gateway via SSH using the default IP address, username, and password shown in the image 4.30. Now, two configuration changes must be made. First, the Gateway must have active Internet access, which can be accomplished by Ethernet cable or by connecting to WiFi. Second, configure the gateway by selecting TTN as the LoRa server and the appropriate frequency range. Finally, we can make a note of the Gateway ID, which we will use during the Gateway registration process on TTN 4.31.

Connecting to the Things Network(TTN)

The registration of Gateway on TTN server follows almost the same steps as we had discussed in section 4.4.1, the only changes are as follows:

After log in to TTN website, redirect to console page and click on "GATEWAYS" icon and click on "register gateway" option and fill out the information's like 'Gateway EUI', 'Description', 'Frequency Plan','Router','Location', and 'Antenna placed', the 'Gateway ID' is obtained from the previous steps as shown in this image 4.33, then click 'Register Gateway' the status will turn to connected which indicates the LoRA-Gateway is up and running as depicted in the picture 4.34.

At this moment, both the end-node and the Gateway is up and running, hence the devices will send georeferned payloads to TTN-sever in HEX format further we decode this data into understandable format. The custom payload conversation code is listed in appendix 7.4 and the final parsed data would look like this 4.35

The complete working prototype is shown here 4.36. The disadvantages of this prototype are :

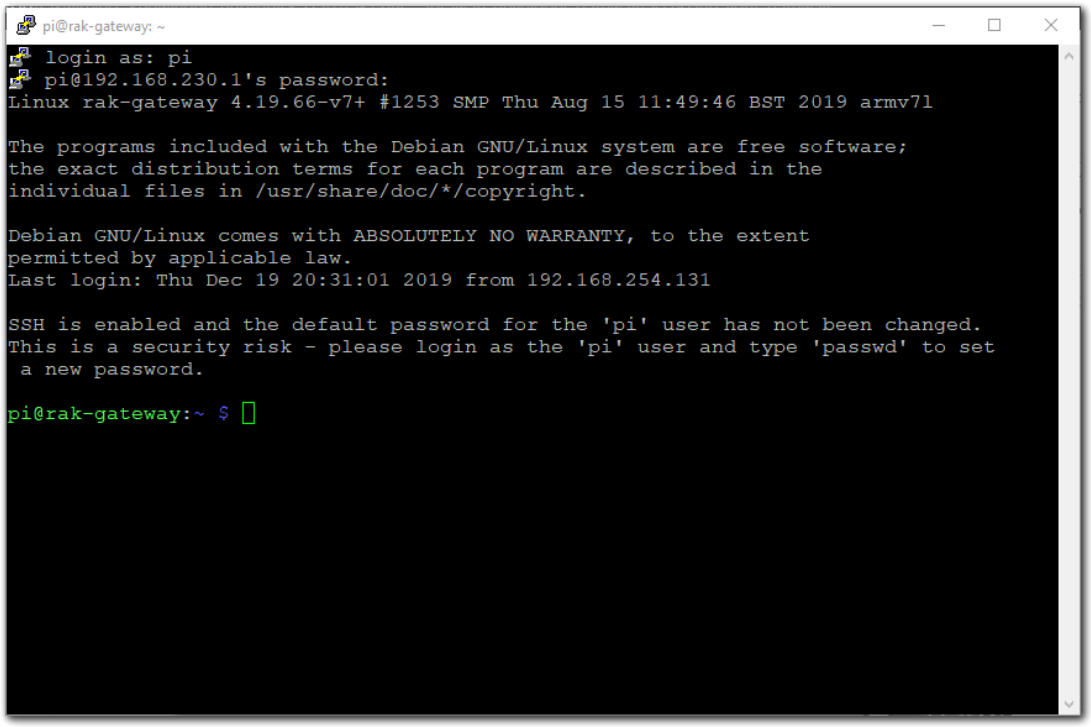


Figure 4.30: Command line after log in to gateway
[Reproduced from docs.rakwireless.com]

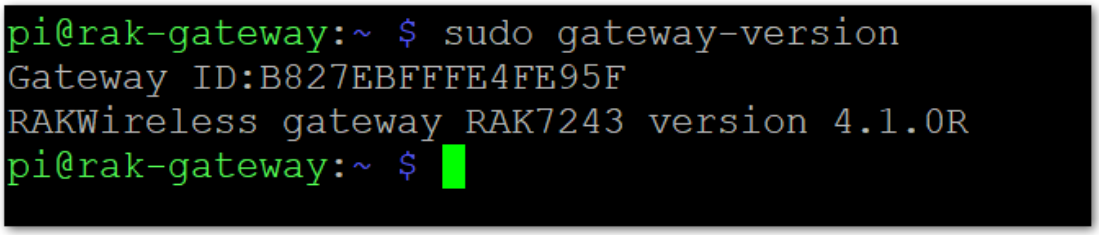


Figure 4.31: LoRa Gateway ID
[Reproduced from docs.rakwireless.com]

1. Low bandwidth, not suitable for real-time application
2. Can only send small packets every few minutes
3. Not idle for continuous monitoring application
4. The data rate is slow due to a radio frequency interference problem.
5. Since the LoRa-Gateway has a weak signal at times, a large number of packets are misrouted and never reach the server.

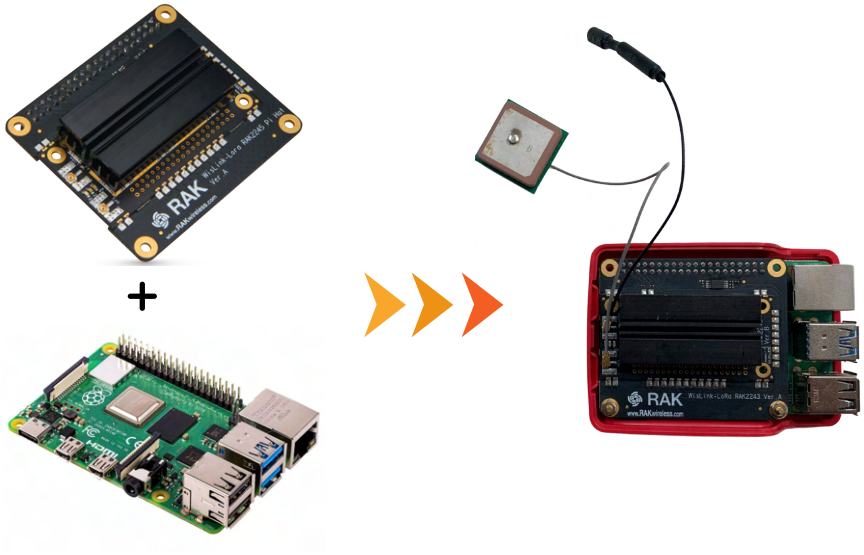


Figure 4.32: LoRa Gateway Module Setup
[Reproduced from docs.rakwireless.com]

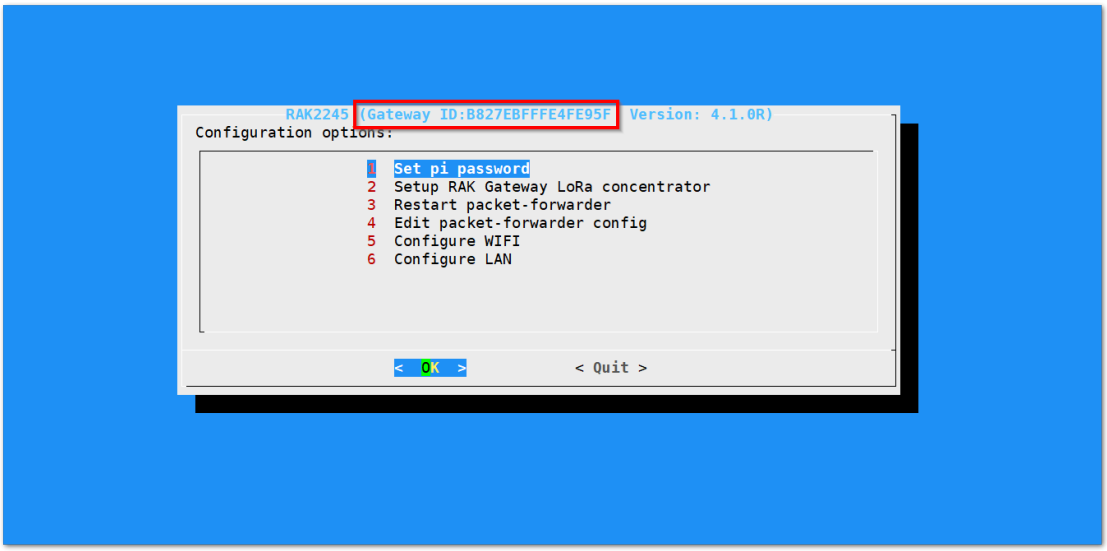


Figure 4.33: RAK2245 Pi Hat Gateway ID in SSH
[Adapted from www.rakwireless.com] [Reproduced from docs.rakwireless.com]

Since we need a continuous stream of data, this prototype is unsuitable for our use case; however, all of these possible disadvantages are addressed by the final prototype Version 5, which we will discuss in the following section.

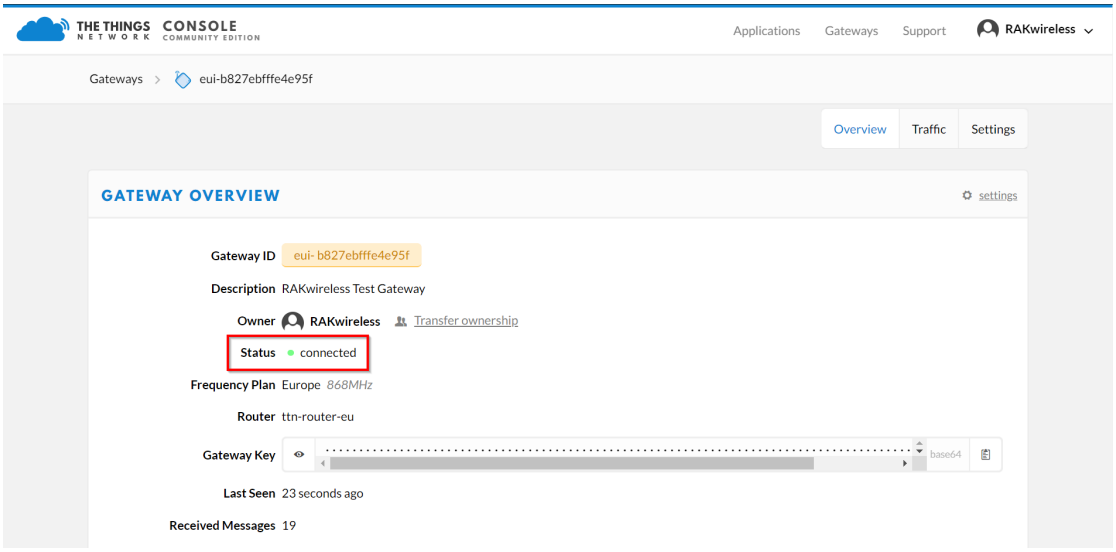
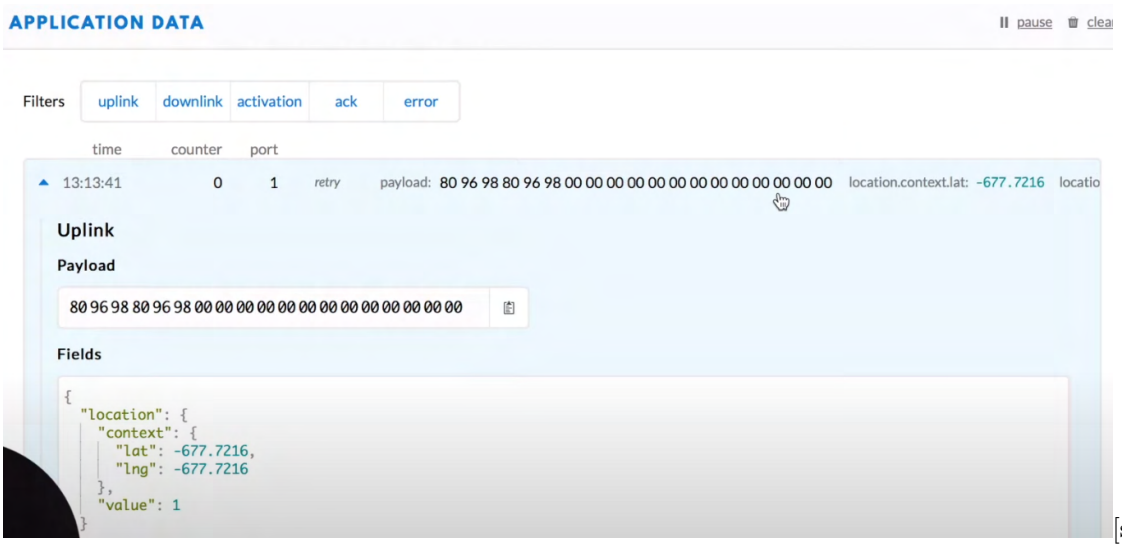


Figure 4.34: RAK2245 Pi Hat TTN Connection Success
[Adapted from www.rakwireless.com] [Reproduced from docs.rakwireless.com]



[source Roadlytics]

Figure 4.35: Final Paylaod in JSON fomrat

4.5 Prototype Version 5 - Final Industry Standard Production Ready Prototype

This is the final prototype that has been sent to production; it was developed and produced by a company named 'ifak' `https://www.ifak.eu/en` and it overcomes all of the drawbacks of the previous prototypes. This system is capable of sending

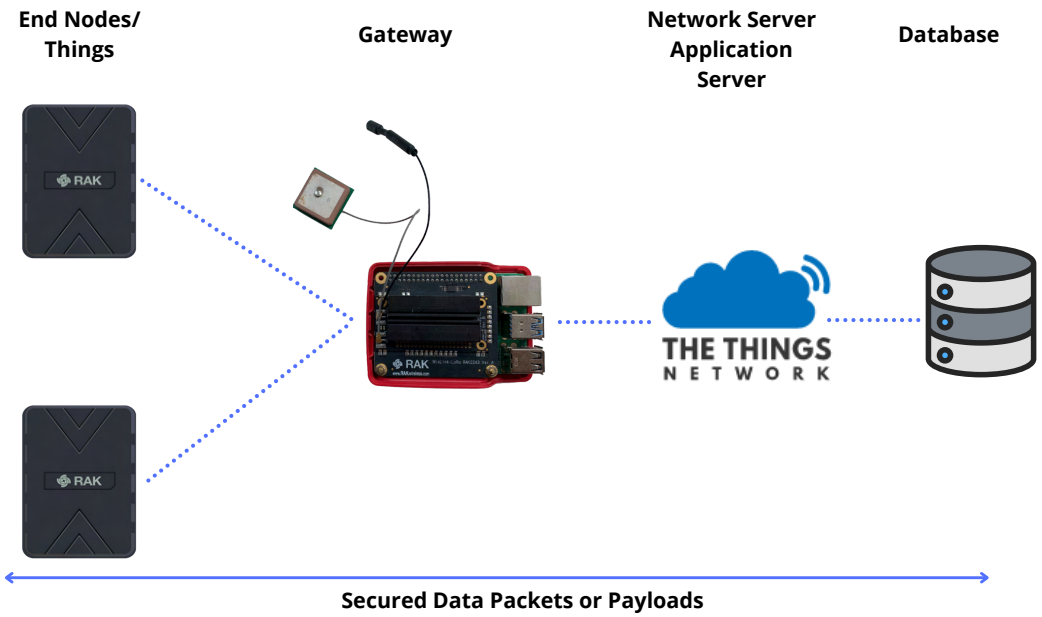


Figure 4.36: Prototype Version 4 with LoRa
[source Roadlytics]

a continuous stream of georeferenced data to the roadlytics server every second in real time, and it uses LTE-M for data transmission. This device is equipped with a Li-Ion battery as a secondary source of uninterrupted power supply and an external power supply plug as a primary source of power supply. We will soon be testing these prototypes in the city of Hamburg in the real environment, but for the time being, a few sample tests have been conducted and the prototypes are working as expected.

4.5.1 This prototype is made up of the following hardware modules

The PCB is custom-built to meet the needs of the Roadlytics-Project, and it includes the following hardware modules.

1. LTE-M modem and MCU: The Nordic nRF9160 Micro Controller Unit features a powerful 64 MHz Arm Cortex-M33 CPU with on-chip 1MB flash and 256 KB RAM dedicated to device use, as well as a multimode LTE-M/NB-IoT modem, RF front End (RFFE), power management, and built-in Assisted GPS. It uses cloud location data in conjunction with GPS satellite trilateration to allow remote monitoring of the device's position. The chip also has built-in cryptographic and security features, enabling the nRF9160 to follow the most current internet security and authentication standards. For connection and authentication, the chip supports both SIM and eSIM. It is a low-power System-in-Package (SiP) with an integrated LTE-M modem and GPS [21].

GNSS		GPS GLO GAL BDS	GPS GLO GAL	GPS GAL	GPS GLO	GPS DBS	GPS
Acquisition	Cold Start	25 sec	25 sec	30 sec	26 sec	28 sec	30 sec
	Hot Start	2 sec	2 sec	2 sec	2 sec	2 sec	2 sec
	Aided Start	2 sec	2 sec	2 sec	2 sec	2 sec	2 sec
Nav. update rate	RTK	8 Hz	10 Hz	15 Hz	15 Hz	15 Hz	20 Hz
	PVT	10 Hz	12 Hz	20 Hz	25 Hz	25 Hz	25 Hz
	RAW	20 Hz	20 Hz	25 Hz	25 Hz	25 Hz	25 Hz
Convergence time	RTK	<10 sec	<10 sec	<10 sec	<10 sec	<10 sec	<30 sec

Table 4.5: ZED-F9P performance in different GNSS modes

2. Dual Frequency GNSS Receiver: The ZED-F9P is a high precision GNSS module by Ublox with integrated multi-band dual frequency Real Time Kinematics (RTK) technology for centimeter level accuracy, the module is capable of concurrent GNSS receiver that can receive and track multiple GNSS constellations like GPS, GLAONASS, Galileo and BeiDou, it has fast convergence time and reliable performance as mentioned in the table 4.5.

3. Dual Frequency GNSS Antenna: The ublox ANN-MB multi-band active GNSS antennas are designed for high precision GNSS applications that require highly accurate location capabilities. It has a compact design, an excellent price-performance ratio, and a variety of mounting and connector options for fast, easy, and reliable multi-band antenna solutions. The ANN-MB antenna is an ideal complement to the new ublox F9 platform, which includes the ZED-F9P module [22].

4. Acceleration and Gyro sensor: The LSM9DS1 is a System-in-package that includes a 3D magnetic sensor, a 3D digital linear acceleration sensor and a 3D digital angular rate sensor, it gives 16-bit data output and has SPI and I2C serial interfaces, it operates with analog voltage supply between 1.9V to 3.6V, some of its feature are the sensor will be "Always-on" mode known as eco power mode, it has both position and motion detection functions along with embedded temperature sensor, it has intelligent power saving technology [23].

5. Power Management Sensor: The LTC2954 governs device power via push-button interface, it has the following features Adjustable Power on or off timers, an input voltage ranges between 2.7V to 26.4V to accommodate a broad range of input power supplies, very low quiescent current (6MuA typical) makes it suitable for battery driven applications [24].

4.5.2 Roadlytics IoT platform

We use Node-Red as our IoT platorm specially for device communication and device management, Node-RED is a programming tool for connecting hardware

devices, APIs, and online services in novel ways. It includes a browser-based editor that makes it simple to link flows using the palette's broad range of nodes, which can be deployed to its runtime with a single click.

1. Device Handshake: When the device is powered on, it shall acquire the following parameters: Firmware version, IMEI, Latitude and Longitude. The firmware version is required to know if there is a firmware upgrade or not. IMEI is for uniquely identifying each device. Latitude and Longitude are to determine the location of the device which will be required to provide the right correction data source. Using these parameters, an HTTP request would be sent to server and waits for a response back that includes control parameters or configuration parameters as shown in this section 4.5.2.

2. Device Management and Communication: When the Roadlytics server recognises the device during the handshake process, it approves that device to send the data with the new updated configuration parameters, the devices send data to the server or database using Message Queuing Telemetry Transport (MQTT) protocol, its a lightweight, publish-subscribe network protocol for transporting messages between devices. MQTT is scalable enough to bind millions of IoT devices. The following node red flow 4.37 handles the device communication process, a mosquitto MQTT broker has deployed [25] on the Azure Virtual Machine where the Node Red is running.

   Lets see how the node-red flows works step by step.

   - Devices publish binary data to a specific MQTT topic.
   - A custom function node will process and parse the binary data to json data.
   - finally, the data is inserted in MongoDB.

```
HTTP REQUEST:
GET /rl/query?fw=256&imei=352656100529406&lat=52.1423159650&
lon=11.6578035350 HTTP/1.0

HTTP RESPONSE:
txPos=1&rxConf=10&useBin=true&useRTK=true&fw-upgrade=0&
sapID=1&appUrl=10.65.163.255& appP=1882

txPos: sending interval of the position measured in seconds.
rxConfig: reading of configuration parameters interval. Measured in minutes.
useBin: use binary format or JSON for MQTT messages.
useRTK: use RTK correction data or only raw position from GNSS.
fw-upgrade: require firmware upgrade or not.
sapID: for connecting to the dedicated correction data source.
appURL: here is the IP of the MQTT broker which will receive the messages from sensor.
appPort: the MQTT broker port.


# Binary data from the device
Binary Data = [1,0,1,10,39,37,254,148,153,37,189,64,1,0,172,203,99,
```

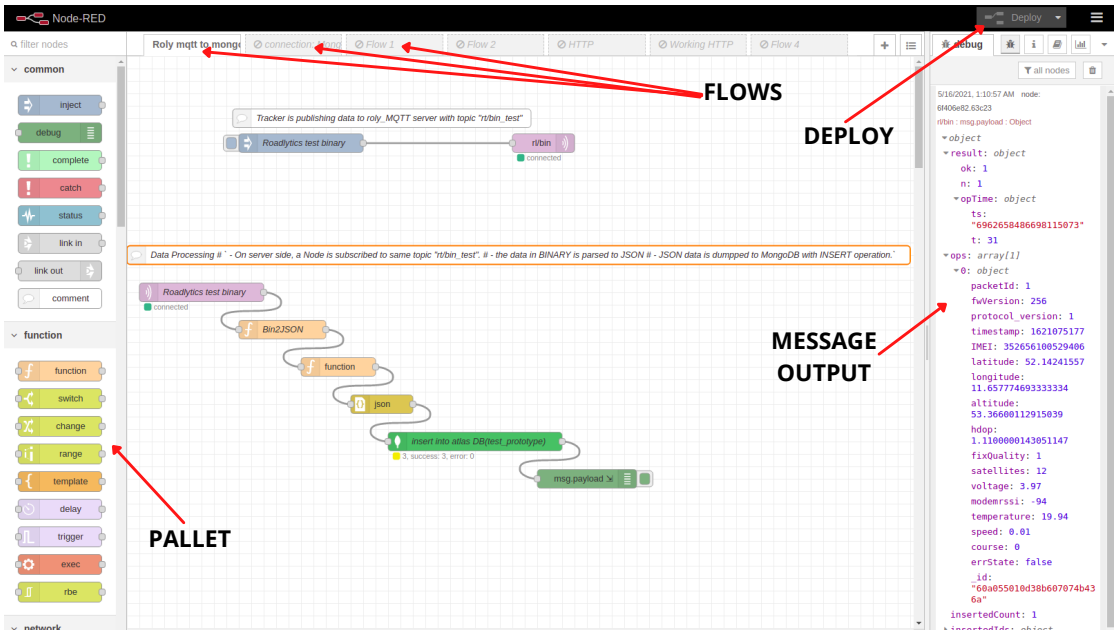


Figure 4.37: Node Red Flow
[source Roadlytics]

```
172,58,18,74,64,150,1,56,216,199,80,39,64,201,118,85,66,123,20,142,
63,1,12,196,221,125,64,94,0,128,159,65,0,111,18,3,60,0,0,192,127,1,0]

# Parsed JSON data

{
    "result": {
        "ok": 1,
        "n": 1,
        "opTime": {
            "ts": "6962658486698115073",
            "t": 31
        }
    },
    "ops": [
        {
            "packetId": 1,
            "fwVersion": 256,
            "protocol_version": 1,
            "timestamp": 1621075177,
            "IMEI": 352656100529406,
            "latitude": 52.14241557,
            "longitude": 11.657774693333334,
            "altitude": 53.36600112915039,
            "hdop": 1.1100000143051147,
            "fixQuality": 1,
            "satellites": 12,
            "voltage": 3.97,
            "modemrssi": -94,
```

```
            "temperature": 19.94,
            "speed": 0.01,
            "course": 0,
            "errState": false,
            "_id": "60a055010d38b607074b436a"
        }
    ],
    "insertedCount": 1,
    "insertedIds": {
        "0": "60a055010d38b607074b436a"
    }
}
```


4.5.3 Roadlytics System Architecture

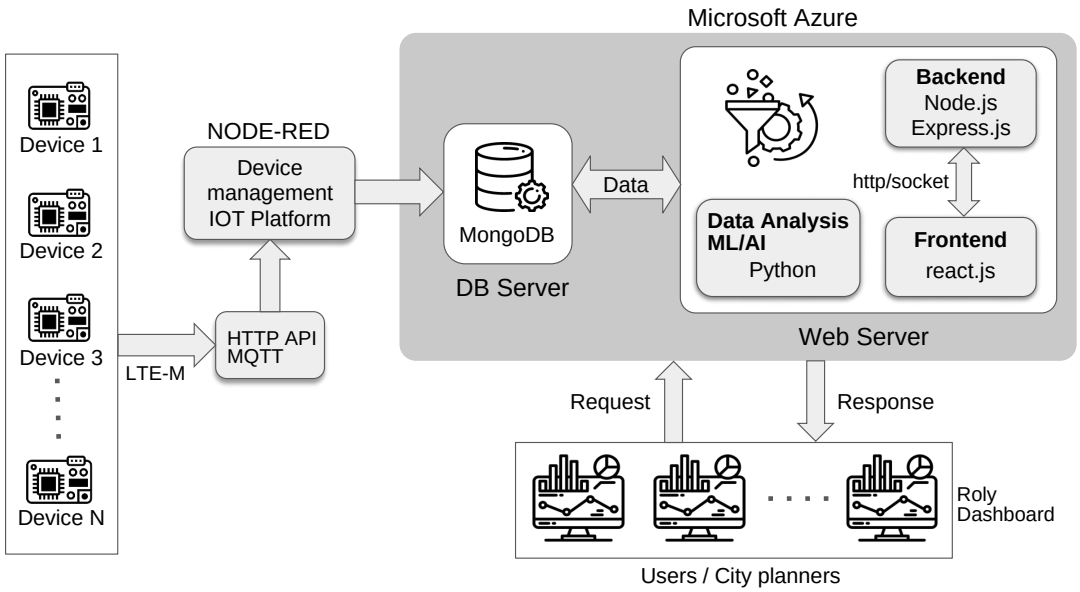


Figure 4.38: High level System Architecture of Roadlytics

The figure 4.38 shows a high level system architecture along with the data flow between the devices and dashboard, the general working of the system is as follows

- IOT devices will send Geolocation data every second to its IoT platform.
- The data is validated, parsed and processed in the IoT platform using Node-Red.
- finally, all the device data is stored into MongoDB collections.
- When user request specific data from dashboard, a backend server spins up and get the required data from MongoDB and displays it onto the screen.

- When the user queries for Hotspots from Roly Dashboard, ML algorithm spins up and process the required data.
- User can finally visualize these data and can make decisions based on the results

4.5.4 Roadlytics Web App

Here are some screenshots of Roadlytics Web App 4.39, which is currently deployed on Microsoft Azure. It has some basic functionalities like date-time selections, hotspot info, signal info and device management section. It is still under development by other software engineer of the Roadlytics team.

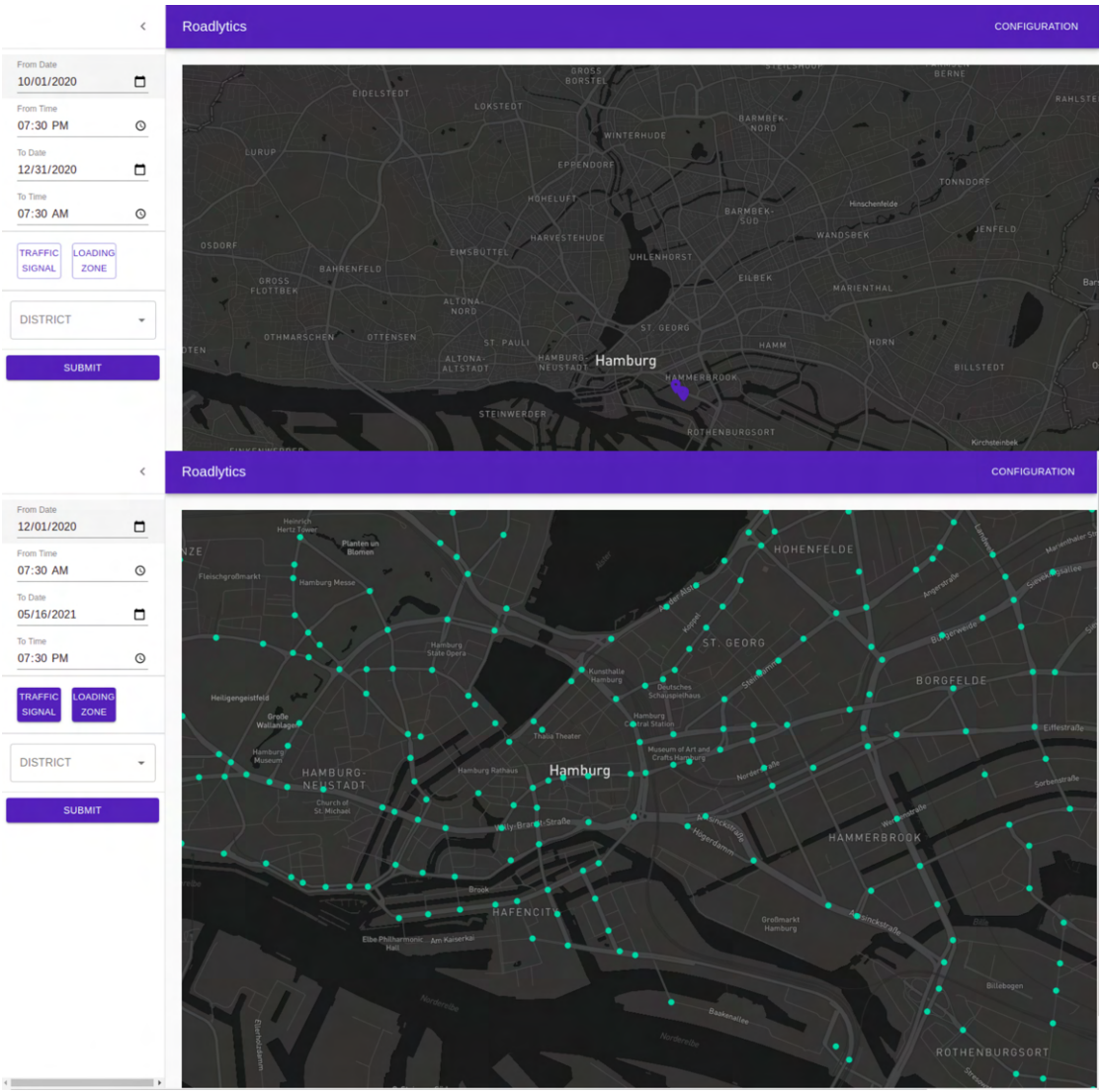


Figure 4.39: roadlytics WebApp Screenshots
[source Roadlytics]

Chapter 5

ML Implementation and Data Analysis

Clustering is a technique of grouping set of data points in a way that similar data points are placed together. Therefore, clustering algorithms look for similarities or dissimilarities among data points. It is an unsupervised learning method so there is no label associated with data points. The algorithm tries to find the underlying structure of the data.

5.1 DBSCAN-*Density Based Spatial Clustering with Noise by sklearn*

DBSCAN is a popular unsupervised learning method utilized in model building and machine learning algorithms. The term ***Unsupervised learning*** means there is no clear objective or outcome we are seeking to find. Instead, we are clustering the large spatial datasets together based on the density of observations. The amazing feature of DBSCAN is it automatically detects and separates outliers and also does not requires to mention number of cluster in advances as in the case of K-Means clustering. The two important parameter of DBSCAN are ***epsilon*** and ***Minimum Points***

- ***epsilon(ϵ):*** This is also referred to as 'eps', it is the distance till which we will look for neighboring points.
- ***Minimum Points(minPts):*** The minimum number of points specified by the user.

Based on these two parameters, points can be classified as core-points, border-points and outlier which is demonstrated in the fig 5.1

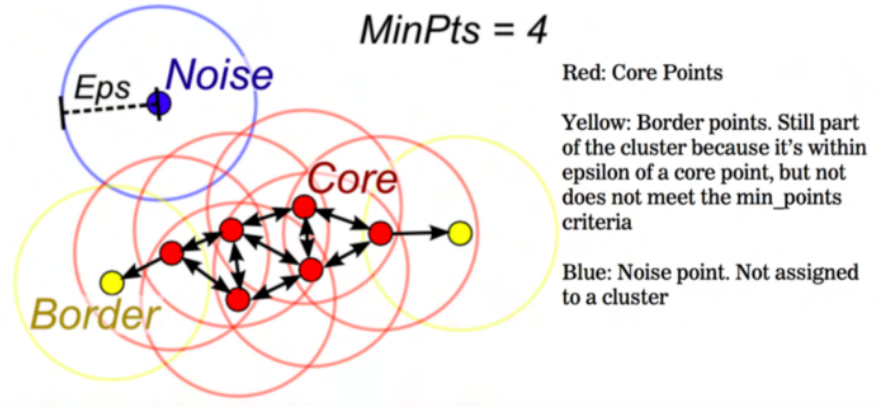


Figure 5.1: Understanding DBSCAN pictorially
[reproduced from medium.com]

- ***Core points:*** A point is said to be core point if it has atleast minimum number of points (minPoints) including itself within its epsilon neighborhood. In fig 5.1, red points are the core points that have atleast minPoints is equal to 4 within epsilon space, note: if we have core points that means it is a dense region
- ***Border points:*** Still part of the cluster because it's within epsilon radius of a core point, but does not meet minPoints criteria. In fig 5.1, yellow points are represented as border points
- ***Noise or outliers:*** A point which is not a core point nor a border points and may not be able to reach from any core points. In fig 5.1, blue points are outlier or noisy points

By performing DBSCAN clustering on trackers data, we were able to achieve high performance throughput by eliminating outliers and by getting the precise centroids of the clusters which known as hotspot. According to our usecase the term ***HOTSPOT*** refers to a precise georefernced position exactly where the delivery vehicle stopped for certain amount of time for delivering the parcels and which could hinder the traffic flow. Knowing where exactly these hotspot will help the city planners to manage traffic jams or to come up with a new project where they can assign dedicated parking zones for logistics vehicles without disturbing the traffic flow.

What is a Hotspot?

In Roadlytics, the term **Hotspot** referes to a stop made by logistic vehicle in order to deliver a shipment to the customer, as a first step, we use machine learning models to detect valid stops in the data, below is a pictorial representation to actual data versus detected stops in data using clustering algorithm.

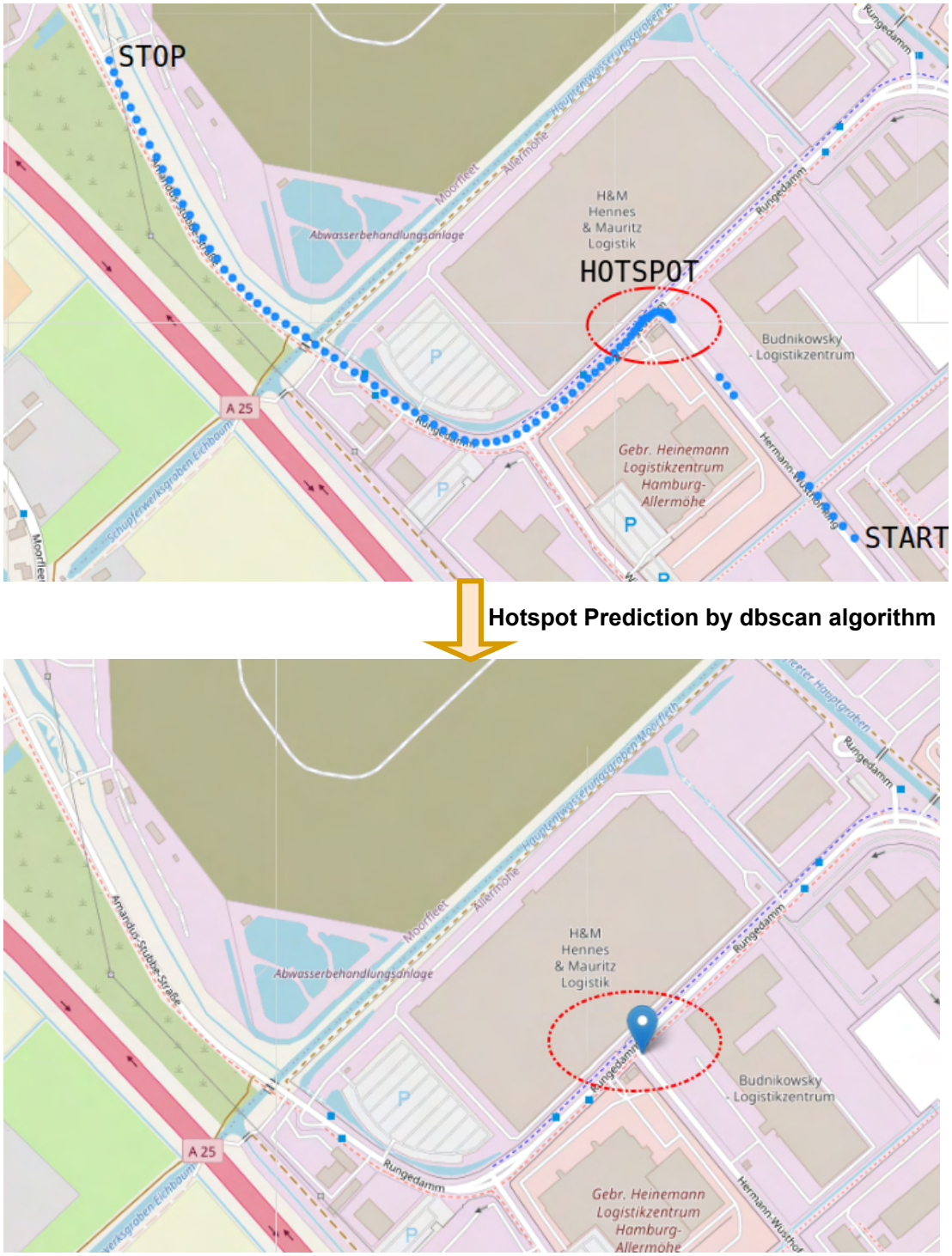


Figure 5.2: predicted hotspot from dbscan and plotted on folium map

In the figure 5.2, The top image contains a test track with labels START and STOP, there is a density area associated in the data, which is highlighted with

red eclipse, this inference is purely based on the visual characteristics of the data, This data was fed into the dbscan model, which predicted the correct hotspot as shown in the image below.

5.1.1 Steps involved in DBSCAN Algorithm

1. The algorithm begins with a random point in the dataset that has not yet been visited, and its neighbors are identified using the ϵ value.
2. If the point has more or equal points than the minPts, the cluster formation begins and this point becomes a "core point"; otherwise, it is considered noise. It is important to note that a point that was initially classified as noise can later become a border point if it is within the ϵ radius of a core point.
3. If the point is a core point, all of its neighbors join the cluster. If the points in the neighborhood turn out to be core points, their neighbors will also be included in the cluster.
4. Repeat the steps above until all points are classified into different clusters or noises.

This algorithm works well if all the clusters are dense enough, and they are well separated by low-density regions. DBSCAN is a simple but powerful algorithm that can classify any number of clusters of any form, is resilient to outliers, and only needs two hyper parameters (ϵ and minPts). However, if the density of the clusters varies greatly, it may be difficult to accurately capture all of them. Furthermore, its computational complexity is approximately $O(m \log m)$, making it nearly linear in terms of the number of instances and space complexity is $O(n)$.

5.1.2 Evaluating Cluster performance

Since the clustering algorithm does not use ground truth labels, validating it is more complex than validating a supervised machine learning algorithm. The ***Silhouette Coefficient*** metric would be useful in our usecase to see how the cluster performs on the given data, allowing us to validate the cluster output more effectively.

- ***Silhouette Coefficient***

  The Silhouette Coefficient is determined for each datapoint using mean intra-cluster distance (x) and the mean nearest-cluster distance (y). Silhouette Coefficient is given by $(y - x)/max(x, y)$. To be more specific, 'y' is the distance between a datapoint and the nearest cluster in which the datapoint does not belong [26]. The Silhouette Coefficient ranges from +1 (the best possible value) to -1. (worst possible value).

- values close to zero indicate the presence of overlapping clusters(Overfit).
- Negative values usually indicate that a sample was assigned to the incorrect cluster while another cluster is more similar (Underfit).

- Values greater than zero mean that the clustering was effective and that the fit is optimal.

DBSCAN- analysis and result									
Original Datasize	No. of Datapoints	Test Cases	Hyperparameters Setup	Total Clusters Formed	Compression Rate	Processing Time	Estimated Noise / Outliers	Silhouette Coefficient	Feedback
114 KB	919 points	1	eps = 0.0001 minPts = 4	16	98.3 %	0.08 sec	175 points	0.549	Optimal
		2	eps = 0.0001 minPts = 6	10	98.9 %	0.02 sec	249 points	0.397	Optimal
		3	eps = 0.0001 minPts = 8	10	98.9 %	0.06 sec	266 points	0.368	Optimal
1.1 MB	6674 points	1	eps = 0.0005 minPts = 6	46	99.3 %	0.34 sec	1321 points	0.085	Overfit
		2	eps = 0.0001 minPts = 4	63	99.1 %	0.33 sec	5462 points	-0.349	Underfit
		3	eps = 0.0001 minPts = 6	56	99.2 %	0.27 Sec	4779 points	0.322	Optimal

Figure 5.3: Model Performance $\Rightarrow SilhouetteScore(underfit, overfitandoptimalfit)$ [source Roadlytics]

Figure 5.3 depicts dbscan model output with varying data sizes and hyperparameter tuning, the 'feedback' column indicates whether the model was underfit, overfit, or optimal fit. Two of such cases are highlighted in red in the table. one characterized by overfitting and the other by underfitting with negative silhouette score. Two of these cases are highlighted in red in the table. One is characterized by overfitting, while the other is characterized by underfitting with a negative silhouette ranking. Let's look at each case with graphical output in the following section.

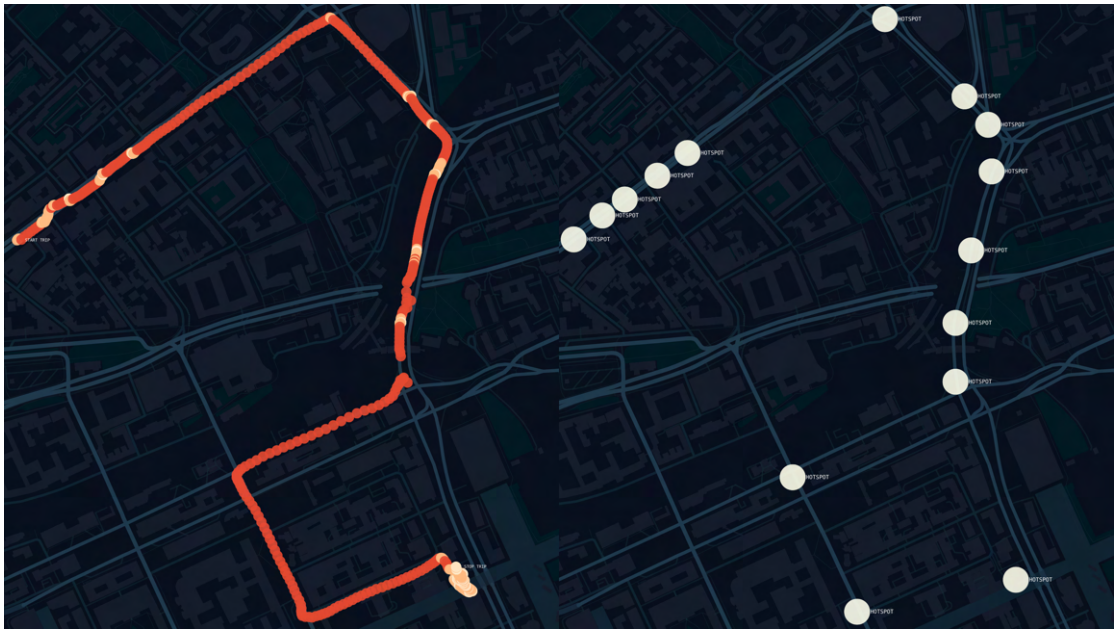
5.1.3 DBSCAN Analysis on tracker dataset 1

The follwing dataset has been retrieved from one of our prototypes tracker which were deployed in testing vehicle. The size of the data is 114 KB with 919 geospatial points which includes latitude and longitude, we ran dbscan on the following dataset to cluster the density points and to find out the hotspots where the van had stopped for delivering the packages. This process carried out by iterations varying by tuning eps and minPts.

Test Case 1

The chosen epsilon is 0.0001 KM, the epsilon distance is in 'Haversine distance,' and the minimum points are set to 4, the model has compressed 98.3 percent of the

original data to shape 16 clusters in 0.08 seconds, and found 175 points as noise or outliers, the silhouette score is 0.549, and the model is considered Optmial fit.



left image: original data right image: Hotspots

Figure 5.4: Test Case 1: eps = 0.0001 and minPts = 4 $\Rightarrow$ $OptimalFit$
[source Roadlytics and kepler]

In the figure 5.4, the image on the left shows the original data filtered with the 'speed' attribute, here the certainty is that when the speed is low, the gps points tend to be closely associated with each other, forming density points(shown with yellow area), and when the speed is high, the points are loosely associated with each other.

When this data is passed to dbscan, the model computes based on the specified hyperparameters and forms a number of clusters, each cluster consisting of density points with minimum speed or speed close to zero, the right image is the final result of dbscan model which only shows Hotspots.

Test Case 2

In this test case, eps was set to 0.0001 km and minPts was set to 6, and the model took 0.02 seconds to compute 10 clusters by compressing 98.9 percent of data with a silhouette score of 0.397, indicating an optimal fit.

The computed hotspots are shown on the right image, and the original data is shown on the left image, plotted on a 3D map. The hotspots are fairly accurate in illustrating the stops made by delivery vehicles.

left image: original data on 3D map right image: Hotspots

Figure 5.5: Test Case 1: eps = 0.0001 and minPts = 6 ⇒ *OptimalFit*
[source Roadlytics and kepler]

Test Case 3

In test case 3, the hyperparameters are set to eps = 0.0001 Km and minPts = 8, the dbscan took 0.06 sec to compute 98.9 percent of data with 10 clusters and a silhouette score of 0.368 representing optimal fit, and the algorithm identified 266 points as outliers.

The hotspots in the above figure 5.6 are depicted by purple dots, which are precisely where the vehicle came to a halt.

Test case comparison on dataset 1

The table 5.1 compares the dbscan algorithm's output on dataset 1 with silhouette metrics score, resulting in optimal fit or the best fit.

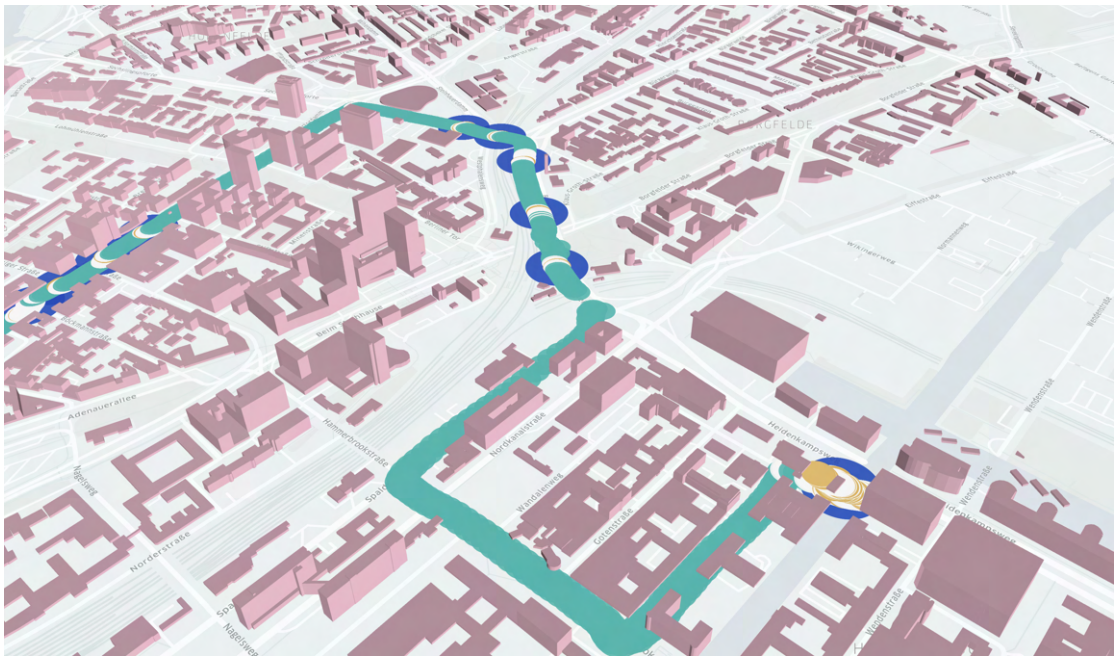


Figure 5.6: Test Case 1: eps = 0.0001 and minPts = 8 ⇒ *OptimalFit*
[source Roadlytics and kepler]

Data Size	114 KB		
No. of Datapoints	919 points		
Hyperparameters Steup	eps = 0.0001 minPts = 4	eps = 0.0001 minPts = 6	eps = 0.0001 minPts = 8
Total Clusters Formed	16	10	10
Data Compression Rate	98.3%	98.9%	98.9%
Processing Time	0.08 sec	0.02 sec	0.06 sec
Estimated Noise / Outliers	175 points	249 points	266 points
Silhouette Coefficient	**0.549**	**0.397**	**0.368**
Feedback	**Optimal Fit**		

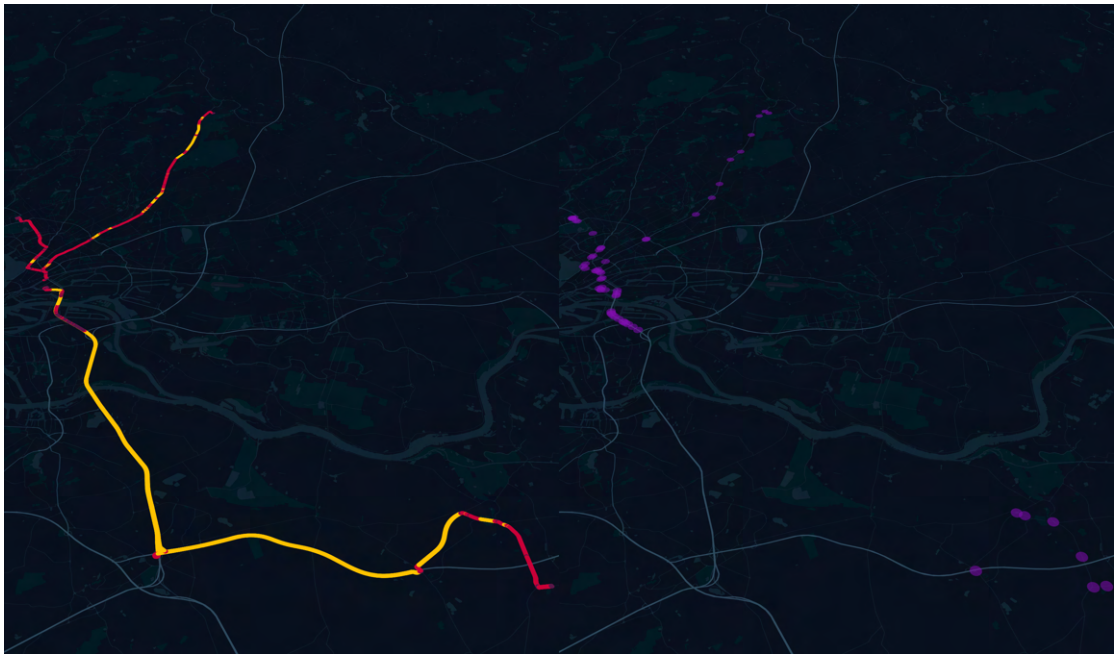
Table 5.1: Comparison of dbscan results with different hyperparameter tuning on dataset 1

5.1.4 DBSCAN Analysis on tracker dataset 2

This dataset was obtained from the above prototypes during the testing process, and it has a data size of 1.1 MB with 6674 georeferenced points. To identify hotspots in this dataset, the above trained dbscan model is used with different hyperparameters as shown in the test cases below.

Test Case 1

In the first test case, the eps value was set to 0.0001 and the minPts value was set to 4, and the algorithm processed 99.06 percent of the data in 0.16 seconds, yielding 63 clusters with silhouette scores of -0.322, as previously mentioned, when the silhouette score is negative, the cluster tends to underfit, which means there vehicle had made lot of stops but the algorithm is unable to detect those stops with the given set of parameters.



left image: original data right image: Hotspots

Figure 5.7: Test Case 1: eps = 0.0001 and minPts = 4 $\Rightarrow UnderFit$
[source Roadlytics and kepler]

In the figure 5.7 The original data is plotted on left image and hotspots are plotted on right image, while validating the analysis dbscan for this case is under performing.

Test Case 2

In this test case, the eps value is set to 0.0005 Km and the minPts is set to 6, and the model produced 46 clusters with 1321 points as outliers within 0.17 sec, generating a silhouette score of 0.085, rendering the clusters over fit as discussed earlies the metric score closer to zero leads to model over fitting. The term over fit refers to creating unwanted hotspots in the dataset, and the right hand image in the figure 5.8 shows the extremely more hotspots than anticipated.

left image: original data right image: Hotspots

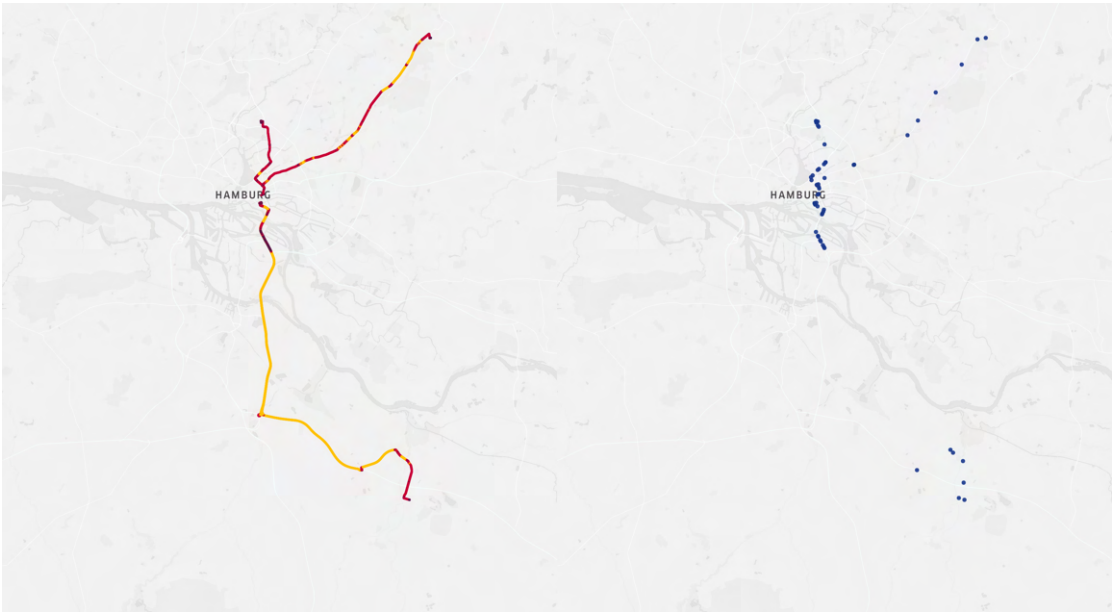
Figure 5.8: Test Case 1: eps = 0.0005 and minPts = 6 $\Rightarrow$ *OverFit*
[source Roadlytics and kepler]

Test Case 3

The final test case leads to optimal fit with the following hyperparameters, the eps value is set to 0.0001 km and minPts is equal to 6, the model took 0.27 sec to process 99.2 Percent of the data forming 56 clusters and neglecting 4779 outliers with the silhouette score of 0.322 which is an optimal fit.

Test case comparison on dataset 2

The following table 5.2 shows the final analysis of the above test cases, with three separate cases where the model performs underfit, overfit, and optimalfit.



left image: original data right image: Hotspots

Figure 5.9: Test Case 1: eps = 0.0001 and minPts = 6 ⇒ *optimalFit*
[source Roadlytics and kepler]

Data Size	1.1 MB		
No. of Datapoints	6674 points		
Hyperparameters Steup	eps = 0.0001 minPts = 4	eps = 0.0005 minPts = 6	eps = 0.0001 minPts = 6
Total Clusters Formed	63	46	56
Data Compression Rate	99.06%	99.31%	99.2%
Processing Time	0.16 sec	0.17 sec	0.27 sec
Estimated Noise / Outliers	4562 points	1321 points	4779 points
Silhouette Coefficient	**-0.322**	**0.085**	**0.322**
Feedback	**Under Fit**	**Over Fit**	**Optimal Fit**

Table 5.2: Comparison of dbscan results with different hyperparameter tuning on dataset 2

5.2 Data Visualization

We use kepler maps as a geospatial analysis tool for the location data to extract insights, it is free and open source project [27], this tool provides various visualization options such as point layer, cluster layer, polygon layer, heatmap layer and many more, the color of the data-points on the map is based on speed factor, when the speed is less than 5Kilo Meter Per Hour (KMPH), teh color is maroon, when the speed is between 5 to 30kmph, the color is red and when the speed is greater than 30kmph, the color is yellow as depicts in the picture 5.10

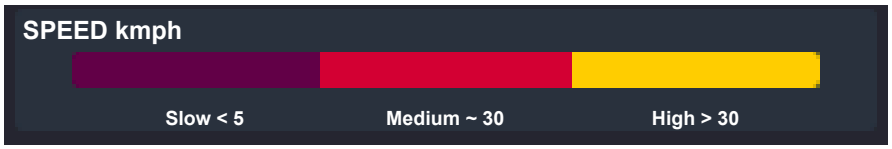


Figure 5.10: Map legends: Speed based on colors [low — Medium — High] [kepler.gl]

In the image below 5.11, the actual data is plotted on kepler map as shown in left pane, as per the color codes, the hotspots are the one which are having 'maroon' points. After passing this data to ML model the predicted hotspots are plotted as green points on the right pane of the image, to compare the accuracy of the predicted hotspots, the actual data is again plotted on top of hotspot data as shown in the left image.

Figure 5.11: Spotting Hotspots [source Roadlytics and Kepler.gl]

The orange points in the picture 5.12, represents traffic signals in the Hamburg. We can see that the density of the data points increases near the signals, which may be due to slow moving traffic near signals or vehicle has stopped because of signal.

Pictures 5.13 and 5.15, shows the entire data from prototypes has been plotted here to show how it looks, and the green dots reflect signals in Hamburg. Based on the density, this visualization shows which are the most common routes taken by vehicles.

Figure 5.12: Density Near Traffic Signal Light
[source Roadlytics and Kepler.gl]

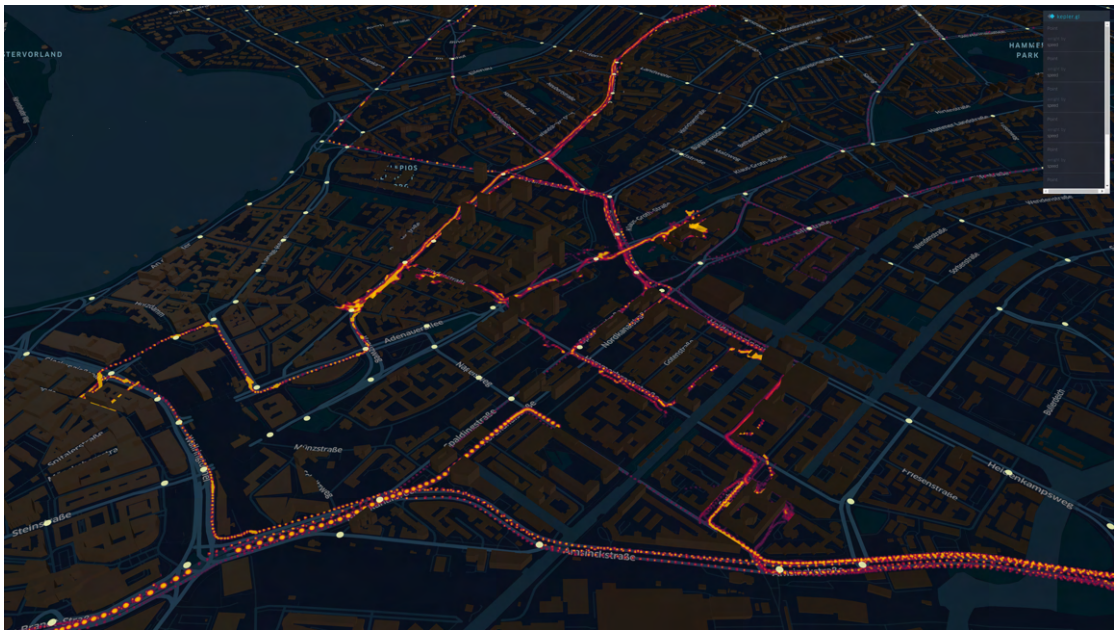


Figure 5.13: Trip Data - 1
[source Roadlytics and Kepler.gl]

The picture 5.14 shows all the traffic signals present in the city of Hamburg.

5.2.1 Clustering hotspot points

The image 5.16, shows the hotspots in clustered form, When the user zooms out, the hotspots cluster as one big point with a number on it showing the total number of hotspot-points, and when the user zooms in, the hotspots scatter.

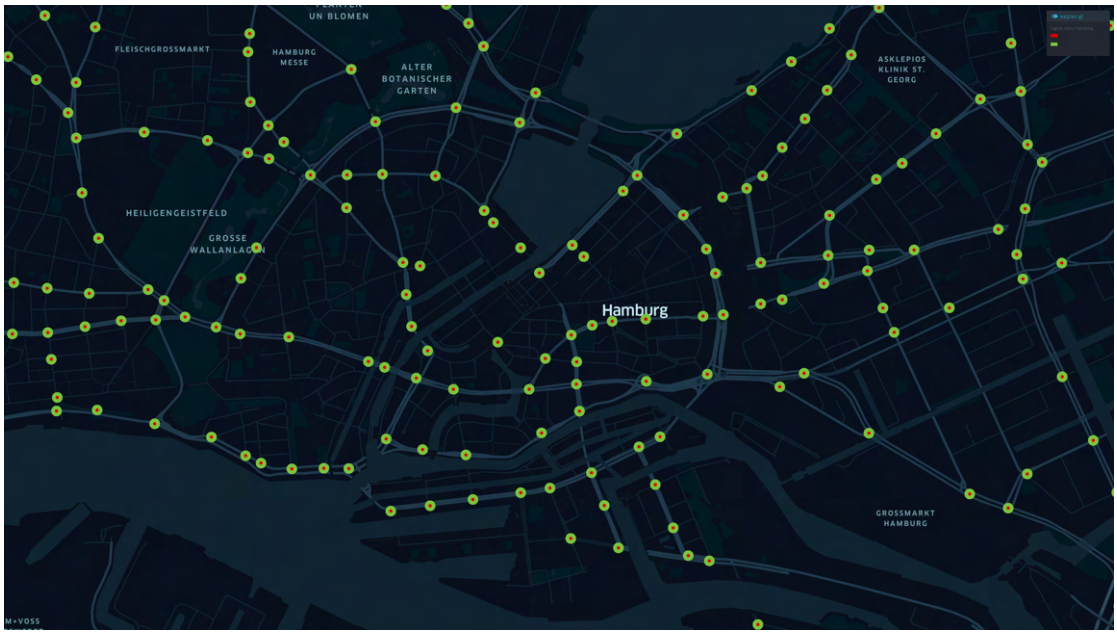


Figure 5.14: Hamburg Traffic Signal Data
[source Roadlytics and Kepler.gl]

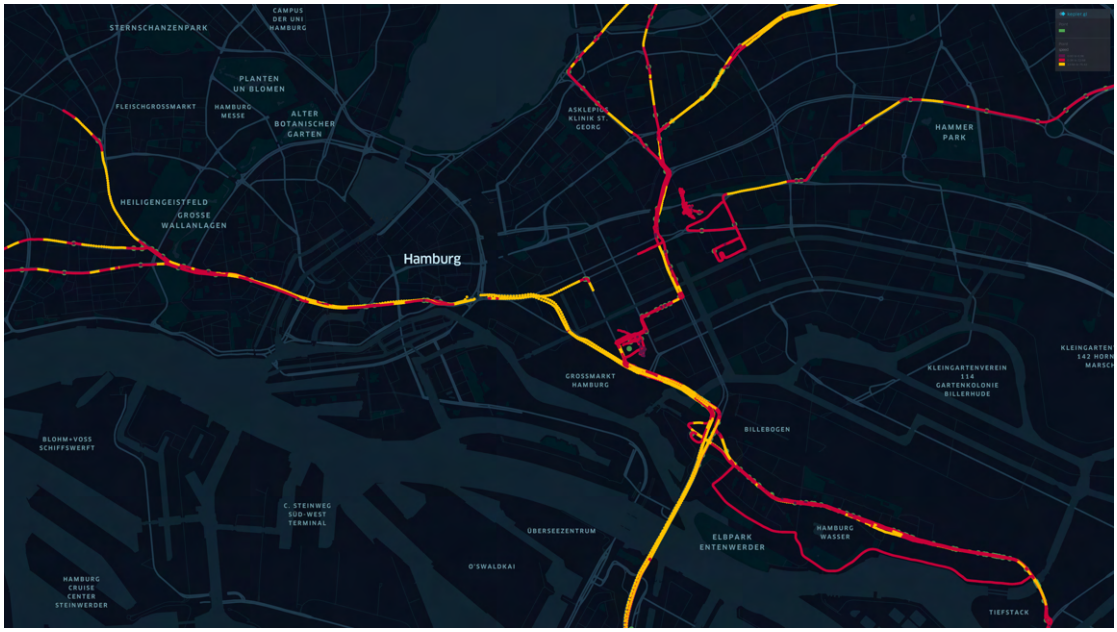


Figure 5.15: Trip data - 2
[source Roadlytics and Kepler.gl]

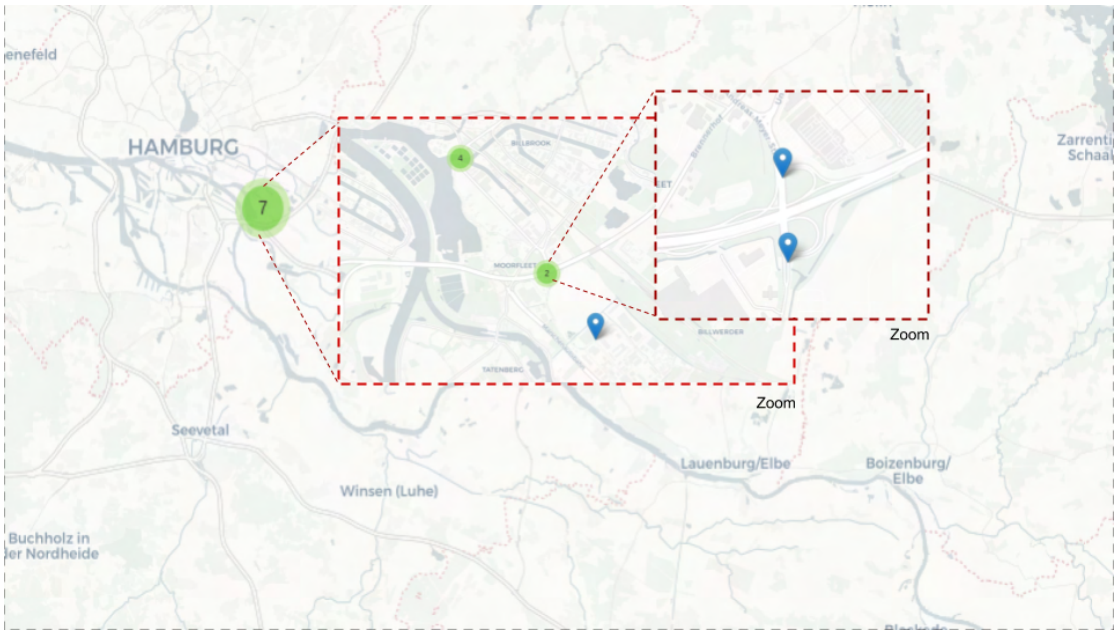


Figure 5.16: Visualization of 7 hotspots on folium maps
[source Roadlytics and python folium maps]

5.2.2 Displaying delivery routes on map

This section shows the delivery routes, the visualization helps city planners to see the most travelled locations by the delivery vehicles in the city of Hamburg.

5.2.3 Displaying data on map to track vehicle stop moments

This section displays the delivery vehicle data where speed was less than 5Kmph, the visualization helps city planners to find out factors affecting the speed of the vehicles, it could be because of traffic jams, or road conditions etc

5.2.4 Displaying Van stopped for delivering

This section shows exactly where and how many times a vehicle stopped for delivering the packages

5.2.5 Displaying data as heatmap

This section shows data on map as heatmap, it helps to understand the most travelled routes of the delivery vehicles as density based.

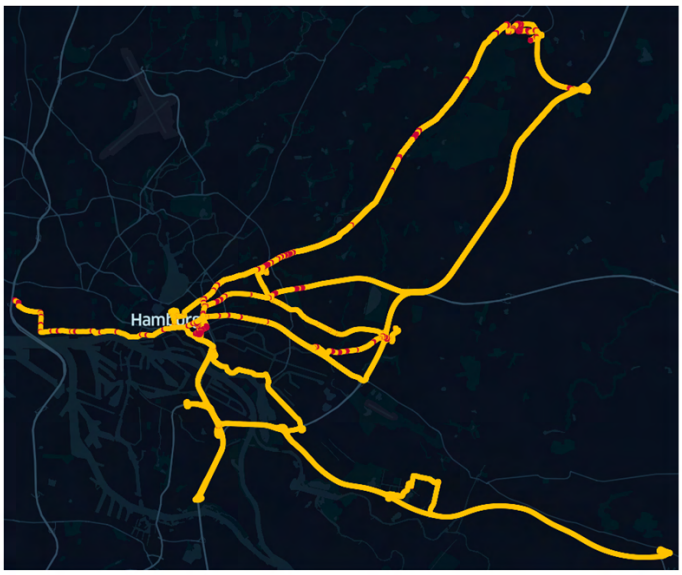


Figure 5.17: Route taken by delivery vehicles - moving data [source Roadlytics and Kepler.gl]

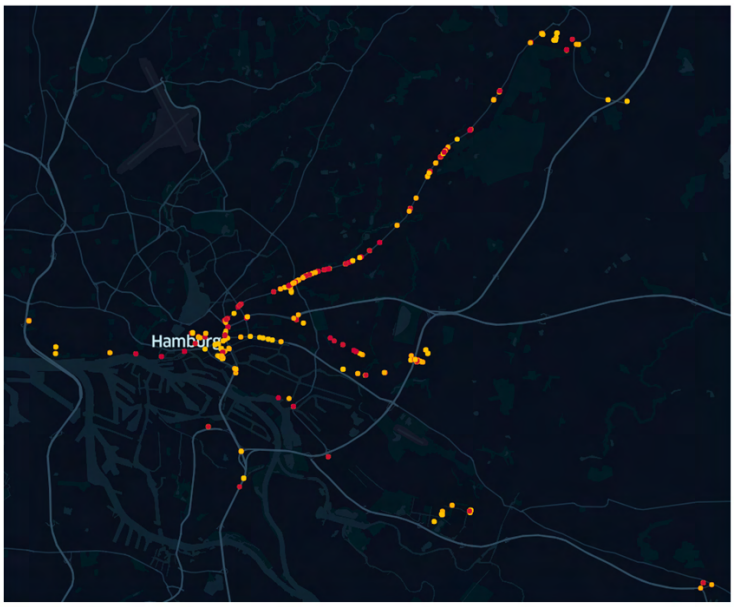


Figure 5.18: Data with speed lessthan or equal to 5kmph [source Roadlytics and Kepler.gl]

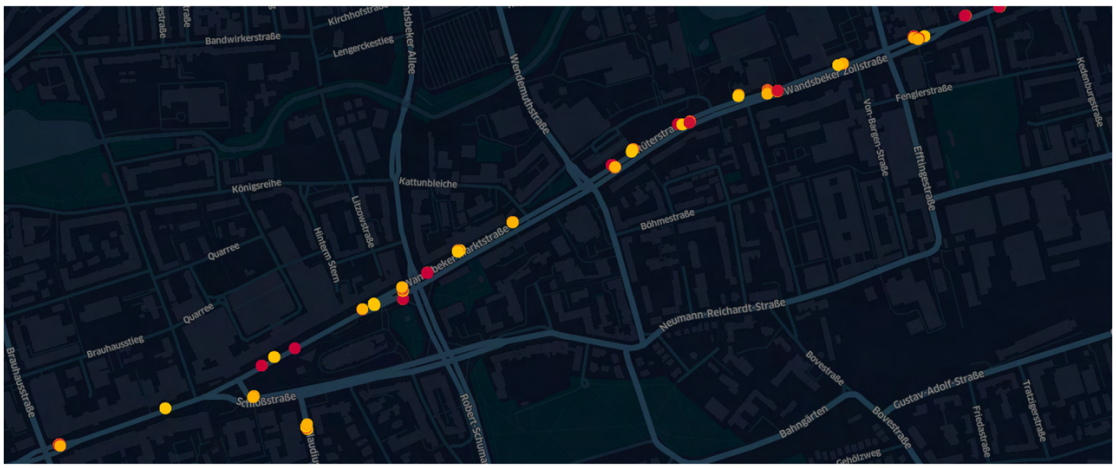


Figure 5.19: Delivery vans stopped for delivering parcels where speed is zero(red dots are basically hotspots)
[source Roadlytics and Kepler.gl]

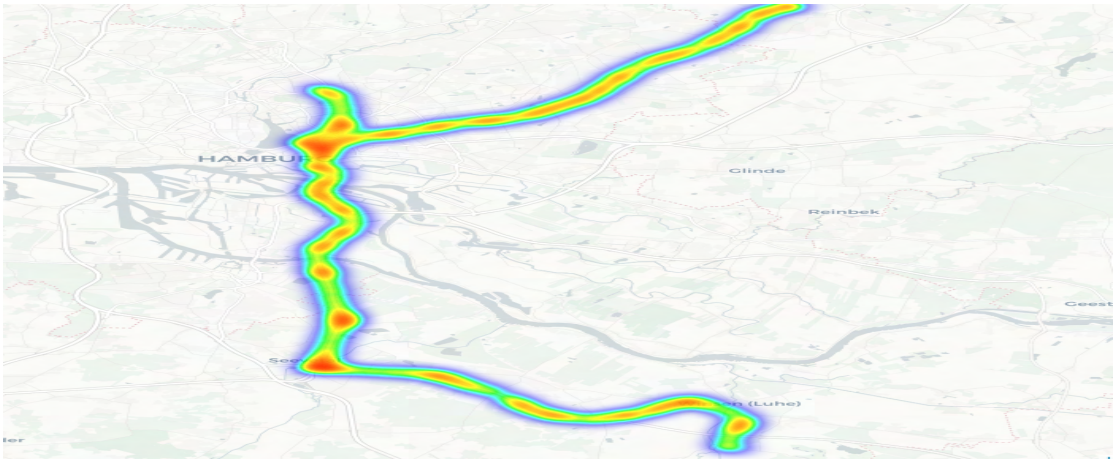


Figure 5.20: Delivery vans frequent routes as heatmap
[source Roadlytics and Kepler.gl]

Chapter 6

Results and Conclusion

6.1 Results and Conclusion

This section talks about the results and conclusion achieved with this project, as stated in the Introduction section, the whole idea was to prototype a working device nothing but GNSS tracker to collect the precise geolocation data as the smart city initiative, the data is further analysed to capture hidden features like finding the hotspots that will help the city planners to make smarter decisions. At this moment there are 6 devices running around hamburg to collect sample data as a pilot phase.

We successfully demonstrated that it is possible to build a commodity tracker device that can be used to generate highly accurate Geo location data, which can then be analyzed using different Machine Learning algorithms to find the exact hotspots where the vehicles parked to deliver the packages; with this, city planners can now know the exact information about the parked vehicles. To successfully complete this project, we used all of the most recent and cutting-edge technology.

6.2 Future Work

LSBG now taking a step forward to upgrade this prototype into a full fledged commercially viable product and they are negotiating with other interested stake holders who are interested in the product, the first version of industry standard prototype has already been in the testing phase, however it is not commercially ready yet. On the other hand we have plans of developing the following

- Universal IoT platform.
- Fully functional Roadlytics platform with all analytical and decision making abilities.
- ML/AI to predict the exact location of the vehicle stopped for example it is on 2nd Row or on bike lane or footpath etc

further more we will incorporate other available data like 'floating car data','road construction data' to make the current system more smarter.

Chapter 7

Appendix

The below is deployed in raspberry pi 4, it is the heart of the tracker which acts as a firmware to solve complex task of receiving GNSS signals, parsing it to JSON readable format and finally saving the data as .log and .json data in file system

7.1 Prototype Version 1 source code

```
import io
import json
import pynmea2
import serial
import datetime
from pathlib import Path

currentTime = datetime.datetime.now()
foldername = "/home/pi/Desktop/Data_GNSS/" + str(currentTime.date()) + "/"
Path(foldername).mkdir(parents=True, exist_ok=True)
ser = serial.Serial('/dev/ttyAMA0', 9600, timeout=5.0)
sio = io.TextIOWrapper(io.BufferedRWPair(ser, ser))
print(sio.readline())
print("here reading it only once for updating the date")
time =  str(currentTime.hour) + str(currentTime.minute) +
        str(currentTime.second)
pushRawData= open(foldername + "rawData_"+ time + ".log","w+")
pushGgaData= open(foldername + "ggaData_"+ time + ".txt","w+")
pushRmcData= open(foldername + "rmcData_"+ time + ".txt","w+")
datagga =[]
datarmc =[]
while 1:
    try:
        line = sio.readline()
        print(line)
        print("------------>")
        msg = pynmea2.parse(line)
        print(repr(msg))
        pushRawData.write(line + "\n")
# pushRawData.writAt the top of the software stack is the application code,
# which makes calls into the middlee()
        if isinstance(msg, pynmea2.types.talker.GGA):
```

```
            print(repr(msg))
            # pushGgaData.write(repr(msg) + "\n")
            latlonggga = {"latitude": msg.latitude, "longitude": msg.longitude}
            json.dump(latlonggga, pushGgaData)
            pushGgaData.write(",")
        if isinstance(msg, pynmea2.types.talker.RMC):
            print(repr(msg))
            latlongrmc = {"latitude": msg.latitude, "longitude": msg.longitude,
            "speed": msg.spd_over_grnd, "date":str(msg.datestamp),
            "timestamp": str(msg.timestamp), "status": msg.status}
            json.dump(latlongrmc, pushRmcData)
            pushRmcData.write(",")
    except serial.SerialException as e:
        print('Device error: {}'.format(e))
        break
    except pynmea2.ParseError as e:
        print('Parse error: {}'.format(e))
        continue
```

7.2 Prototype Version 3 source code - Initial State

```
import gps
import time
# Streaming data and functions to initial states
from ISStreamer.Streamer import Streamer
# parsing NMEA data to lat,lon
from geopy.geocoders import Nominatim
from geopy.exc import GeocoderTimedOut
# print exceptions and traceback
import sys, traceback
from sys import exit

# for Open Street Maps (OSM) Geocoding and reverse Geocoding
geolocator = Nominatim()
# extablish session of the GNSS/GPSS program
session = gps.gps("127.0.0.1", "2947")
session.stream(gps.WATCH_ENABLE | gps.WATCH_NEWSTYLE)
publish_data = Streamer(
bucket_name="GNSS",bucket_key=" ",access_key=" ")
# stream data from raspbery pi to initial satate replace bucket_id and bucket_key
while True:
        try:
            time.sleep(0.1)
            raw_data = session.next()
            if raw_data['class'] == 'GNRMC':
                if hasattr(raw_data,'lat') and hasattr(raw_data,'lon'):
        publish_data.log("Location" ,"{lat},{lon}".format(lat=raw_data.lat,lon=raw_data.lon))
                    print "Latitude is = "+str(raw_data.lat)
                    print "Longitude is = "+str(raw_data.lon)
                    coordinates = str(raw_data.lat) + "," + str(raw_data.lon)
                    where_it_is = geolocator.reverse(coordinates,timeout=10)
                    publish_data.log("Vehicle is located at",where_it_is.address)
                    print(where_it_is.address)
        if raw_data['class'] =='GNRMC':
```

```
        if hasattr(raw_data,'speed'):
        publish_data.log("Speed of the vehicle", raw_data.speed)
                print "Vehicle is moving at = "+str(raw_data.speed)+" KPH"
                if raw_data['class'] =='GNRMC':
                    if hasattr(raw_data,'alt'):
                            publish_data.log("Altitude",raw_data.alt)
                                print "The altitude is = "+str(raw_data.alt)+" m"
                if raw_data['class'] == 'GNRMC':
                        if hasattr(raw_data,'time'):
                                publish_data.log("Time",raw_data.time)
                                print "The curent date and time is = "+str(raw_data.time)+"\n"

        except GeocoderTimedOut as e:
                publish_data.log("msg","Geocoder Timeout")
                pass
        except KeyError:
                pass
        except (KeyboardInterrupt, SystemExit):
                publish_data.close()
                print "\nEnding the current process"
                gps.running = False
                exit()
                quit()
        except StopIteration:
                session = None
                print "No incoming data from the GNSS module"
```

7.3 HTTP API - python flask framework

```
# import main Flask class and request object
from flask import Flask
from flask import request
import warnings
from pymongo import MongoClient
import time
from werkzeug.urls import url_encode
warnings.filterwarnings("ignore")

# connect to the mongoclient
client = MongoClient()

# Load collection name
db = client.get_database("testDB")

# create the Flask app
app = Flask(__name__)


@app.route("/rl/query")
def query():
    fw = request.args["fw"]
    imei = request.args["imei"]
    lat = request.args["lat"]
    lon = request.args["lon"]
```

```
26
27      if db.test_deviceList.count_documents({"imei": imei}) != 0:
28
29          updateDeviceList = db.test_deviceList.find_one(
30              {"imei": imei}, {"_id": 0, "imei": 1}
31          )
32          newValues = {"$set": {"fw": fw, "lat": lat, "lon": lon}}
33          db.test_deviceList.update_one(updateDeviceList, newValues)
34
35          updateLastSeen = db.test_activeDevice.find_one(
36              {"imei": imei}, {"_id": 0, "imei": 1}
37          )
38          newValues_1 = {
39              "$set": {
40                  "lastSeen.timestamp": int(time.time()),
41                  "lastSeen.lat": lat,
42                  "lastSeen.lon": lon,
43                  "status": "active",
44                  "firmware": fw
45              }
46          }
47          db.test_activeDevice.update_one(updateLastSeen, newValues_1)
48
49          imei_val = updateDeviceList.get("imei")
50          activeDevice = db.test_activeDevice.find_one(
51              {"imei": imei_val},
52              {
53                  "_id": 0,
54                  "configParams": {
55                      "txPos": 1,
56                      "rxConf": 1,
57                      "useBin": 1,
58                      "useRTK": 1,
59                      "fw-upgrade": 1,
60                      "sapID": 1,
61                      "appUrl": 1,
62                      "appP": 1,
63                  },
64              },
65          )
66          return url_encode(activeDevice["configParams"])
67      else:
68          print(
69              "The requested IMEI is not found in our database, please register the device first!!!!"
70          )
71
72
73  if __name__ == "__main__":
74      # run app in debug mode on port 5000
75      app.run(debug=True)
```

7.4 Prototype Version 3 source code - Initial State

```
function Decoder(b, port) {


var longitude  = (b[0] | b[1]<<8 | b[2]<<16 | (b[2] & 0x80 ? 0xFF<<24 : 0)) / 10000;

var latitude = (b[3] | b[4]<<8 | b[5]<<16 | (b[5] & 0x80 ? 0xFF<<24 : 0)) / 10000;

return {

location: {
"value":1,
"context":{"lat": latitude, "lon": longitude}
}
};
}
```

7.5 Unsupervised ML model: DBSCAN implementation in python

```
"""
Spyder Editor
@author: Nizam

usage : python3 dbscan_hotspot_detection.py ~/path_to_json_file.json

This is a DBSCAN algorithm to compute Hotspot (Hotspots are the points where
deivery vehciles had stopped for delivering packages)
"""
import numpy as np
import pandas as pd
import sys
from sklearn.cluster import DBSCAN
from pymongo import MongoClient

# to load the computed centroids in mongodb collection by name 'tracker_hotspot'
client = MongoClient("mongodb+srv://")
db = client.get_database('roadlytics_testDB')
collection = db.tracker_hotspot

# Function to insert json data into 'tracker_hotspot' collection
def load_centroids_mongodb(data):
    collection.insert_many(data.apply(lambda x: x.to_dict(), axis=1).to_list())


#function to compute centroids        function Decoder(b, port) {


var longitude  = (b[0] | b[1]<<8 | b[2]<<16 | (b[2] & 0x80 ? 0xFF<<24 : 0)) / 10000;

var latitude = (b[3] | b[4]<<8 | b[5]<<16 | (b[5] & 0x80 ? 0xFF<<24 : 0)) / 10000;
```

```
32
33  return {
34
35  location: {
36  "value":1,
37  "context":{"lat": latitude, "lng": longitude}
38  }
39  };
40  }
41  def get_centroid(cluster):
42      cluster_ary = np.asarray(cluster)
43      centroid = cluster_ary.mean(axis = 0)
44      return centroid
45
46  #final dbscan clustering
47  def dbscan(X, eps, min_samples):
48      db = DBSCAN(eps=eps, min_samples=min_samples, algorithm='ball_tree',
49      metric='euclidean').fit(X)
50      # Training the model
51      #y_pred = db.fit_predict(X)
52
53      #Cluster labels, number of clusters and core points
54      cluster_labels = db.labels_
55      n_clusters = len(set(cluster_labels)) - (1 if -1 in cluster_labels else 0)
56      clusters = pd.Series([X[cluster_labels == n] for n in range(-1, n_clusters)])
57
58      # get centroid of each cluster
59      fac_centroids = clusters.map(get_centroid)
60  #unzip the list of centroid points (lat, lon) tuples into separate lat and lon lists
61      cent_lats, cent_lons = zip(*fac_centroids)
62  # from these lats/lons create a new df of one representative point for eac cluster
63      hotspots_data = pd.DataFrame({'latitude':cent_lats,'longitude':cent_lons})
64
65      load_centroids_mongodb(hotspots_data)
66
67      print("inserted hotspots to mongo collection(tracker_hotspot) successfully")
68      #Calculating NOISE or OUTLIERS in the data points
69      #noise_ = list(cluster_labels).count(-1)
70  #print('Estimated number of NOISE points or OUTLIERS
71  found in the dataset: %d' % noise_)
72      return hotspots_data
73
74  if __name__ == "__main__":
75      if len(sys.argv) < 1:
76          print("Input File Missing!!, Please add your json path")
77          sys.exit()
78
79      with open(sys.argv[1], 'r') as f:
80          data = pd.read_json(f)
81          df = pd.DataFrame(data)
82          X = np.array(df[['latitude','longitude']])
83         # print(X)
84
85      dbscan(X, eps=0.00001, min_samples=3)
86     # load_centroids_mongodb(cent)
```

Bibliography

[1] TomTom, *Hamburg traffic report — tomtom traffic index*, (Accessed on 12/07/2020). [Online]. Available: https://www.tomtom.com/en_gb/traffic-index/hamburg-traffic.

[2] H. City, *Hamburg on the way to becoming a smart city - hamburg.de*, https://www.hamburg.de/bsw/lgv-gremien-projekte/7967698/my-smart-life/, (Accessed on 12/06/2020).

[3] B. eV, *Germany's smartest cities: Hamburg defends the title, munich and cologne are catching up — bitkom ev*, (Accessed on 12/06/2020). [Online]. Available: https://www.bitkom.org/Presse/Presseinformation/Deutschlands-smarteste-Staedte-Hamburg-verteidigt-den-Titel-Muenchen-und-Koeln-holen-auf.

[4] H. SmaLA, *The concept of smala - hamburg.de*, (Accessed on 12/14/2020). [Online]. Available: https://www.hamburg.de/bwi/smarte-ladezonen/13633832/vorgehen/#:~:text=Das%20Projekt%20%E2%80%9ESmaLa%E2%80%9C%20wird%20in,Ladezonensituation%20in%20Hamburg%2DMitte%20gemacht..

[5] H. City, *Its projects in hamburg - hamburg.com*, (Accessed on 12/07/2020). [Online]. Available: https://www.hamburg.com/business/its/11747506/projects/.

[6] L. Hamburg City, *Its - intelligent transport systems*, (Accessed on 12/07/2020). [Online]. Available: https://lsbg.hamburg.de/its-projekte/#anker_4.

[7] K. Tschimmel, "Design thinking as an effective toolkit for innovation," Jan. 2012. [Online]. Available: https://www.researchgate.net/publication/236135862_Design_Thinking_as_an_effective_Toolkit_for_Innovation.

[8] L. Bliss and A. Small, *The startups tracking how cities use curb space - bloomberg*, https://www.bloomberg.com/news/articles/2019-04-02/the-startups-tracking-how-cities-use-curb-space, (Accessed on 07/07/2020), Apr. 2019.

[9] G. T.
bibinitperiod Invest, *E-commerce*, https://www.gtai.de/gtai-en/invest/industries/digital-economy/e-commerce-65482, (Accessed on 07/07/2020), 2020.

[10] E. Morganti, S. Seidel, C. Blanquart, L. Dablanc, and B. Lenz, "The impact of e-commerce on final deliveries: Alternative parcel delivery services in france and germany," in *Transportation Research Procedia*, vol. 4, Elsevier, 2014, pp. 178–190. DOI: 10.1016/j.trpro.2014.11.014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S235214651400297X.

[11] D. DOBERSTEIN, *Gps fundamentals a hardware approach book i of ii*. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/GPS/fundamentals_of_gps_receivers-v1.pdf.

[12] *National marine electronics association - nmea*, (Accessed on 12/17/2020). [Online]. Available: https://www.nmea.org/content/STANDARDS/NMEA_0183_Standard.

[13] *Teach, learn, and make with raspberry pi*. [Online]. Available: https://www.raspberrypi.org/.

[14] *Sam-m8q module — u-blox*. [Online]. Available: https://www.u-blox.com/en/product/sam-m8q-module.

[15] *Installing operating system images - raspberry pi documentation*, (Accessed on 12/17/2020). [Online]. Available: https://www.raspberrypi.org/documentation/installation/installing-images/.

[16] *Jsonlint - the json validator*. [Online]. Available: https://jsonlint.com/?code=.

[17] *Kepler.gl*. [Online]. Available: https://kepler.gl/.

[18] *Initial state - iot platform for data visualizations*. [Online]. Available: https://www.initialstate.com/.

[19] *Li-ion battery hat for raspberry pi, 5v regulated output, bi-directional quick charge, 14500 lithium battery*. [Online]. Available: https://www.waveshare.com/li-ion-battery-hat.htm.

[20] R. V. HENNA DAVID, *Smart irrigation management system using lora wan based sensor nodes*, Nov. 2020. [Online]. Available: https://www.ripublication.com/ijaerspl20/ijaerv15n1spl_13.pdf.

[21] N. Semiconductor, *Nrf9160, low power sip with integrated lte-m/nb-iot modem and gps*. [Online]. Available: https://www.nordicsemi.com/Products/Low-power-cellular-IoT/nRF9160/.

[22] u-blox, *Ann-mb series l1/l2 multi-band, high precision gnss antennas*. [Online]. Available: https://www.u-blox.com/en/product/ann-mb-series#tab-documentation-resources.

[23] *Lsm9ds1 - 9-axis inemo inertial module (imu): 3d magnetometer, 3d accelerometer, 3d gyroscope with i2c and spi - stmicroelectronics*. [Online]. Available: https://www.st.com/en/mems-and-sensors/lsm9ds1.html.

[24] *Ltc2954 datasheet and product info — analog devices*. [Online]. Available: https://www.analog.com/en/products/ltc2954.html.

[25] *Eclipse mosquitto, an open source mqtt broker*. [Online]. Available: https://mosquitto.org/.

[26] *Sklearn.metrics.silhouette_score — scikit-learn 0.24.1 documentation*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html#sklearn.metrics.silhouette_score.

[27] *Github - keplergl/kepler.gl: Kepler.gl is a powerful open source geospatial analysis tool for large-scale data sets.* [Online]. Available: https://github.com/keplergl/kepler.gl.