



Norwegian University of
Science and Technology

A Flexible Way to Dynamically Visualize Spatiotemporal Data in a Geographic Information System

Soma Das

Master of Science in Telematics - Communication Networks and Networked

Submission date: July 2018

Supervisor: Peter Herrmann, IIK

Co-supervisor: Ergys Puka, IIK
Tomas Levin, Statens vegvesen

Norwegian University of Science and Technology

Department of Information Security and Communication Technology



NTNU – Trondheim
Norwegian University of
Science and Technology

A Flexible Way to Dynamically Visualize Spatiotemporal Data in a Geographic Information System

Soma Das

Submission date: July 3, 2018
Responsible professor: Peter Herrmann, IIK (NTNU)
Supervisor: Ergys Puka, IIK (NTNU)
Co-supervisor: Tomas Levin, Statens vegvesen

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title: A Flexible Way to Dynamically Visualize Spatiotemporal Data in a Geographic Information System
Course: TTM4905 - Communication Technology, Master's Thesis
Student: Soma Das

Problem description:

The purpose of this thesis is to collect location-specific information from Android devices and to transfer that data to a remote server where it will be stored in a database (SQL,MYSQL,PostgreSQL, etc.). Moreover, the server will have the capability to analyze the data and to transform them into a format that makes the easily understandable visualization on maps possible. The amended data will be forwarded to the open source geographical information system, QGIS where they can be displayed on moving maps in real time. Further, the server respectively QGIS shall be able to analyze the location-aware data further and to predict certain results. Initially, we will mainly visualize collected cellular network information, but the system shall be scalable and flexible to deal with other kinds of geographic data as well.

The main tasks to be performed in this master thesis are as follows:

- Create a server for storing and analyzing location-specific data and making predictions out of that.
- Allow the reception of data coming from various Android devices by the server.
- Integrate the server with QGIS such that the two systems can communicate in real time.
- Display various geographical patterns using QGIS for the data gathered and stored in the database.

Responsible professor: Peter Herrmann, IIK (NTNU)
Supervisor: Ergys Puka, IIK (NTNU)
Co-supervisor: Tomas Levin, Statens vegvesen

Abstract

We massively make practical and effective use of particular information in our day-to-day lives. We persistently keep an eye on weather pattern which affects what we wear, or traffic pattern information gleaned from various resources. This instinctive need to get from one point to another keeps us constantly in motion and this very ever-changing positions on the globe influence the specifics of what information we almost every day personally consume. So, how do we get along with this volume of data? How to extract the necessary information from the position based on raw data? In reality, it is too complex and simply not possible without help.

Fortunately, we live in an era where we have developed performant ways that allow us to visualize complex data and see important fragments in that information. And one in many ways is Geovisualization using diverse Geographical Information System (GIS) technologies available today.

One such area which is rapidly scaling and require serious attention is the analysis of road traffic data. Data visualization is a graphical approach to discover hidden patterns out from complicated data format such as geospatial or spatiotemporal which contain both time and space qualities and its logical interpretation is not possible with a bare human eye. Besides, it adds valuable aid in the exploration and fetching meaningful conclusion.

In this thesis, we establish an application for the integration of real-time road traffic data with can be dynamically visualized on geographical maps using various attributes present in the data. To this end, we explore various hidden traffic patterns by executing various specific analysis which will eventually be of service in making better decisions in the sector such as road and traffic management or cellular network management.

For this purpose, we choose to work with QGIS, a wide-spread GIS technology which is flexible and user-friendly for Geovisualization purposes. It provides multiple support GIS features to work in both its desktop as well as its server application. Initially, we have worked extensively on both QGIS Desktop platform and QGIS Server which allow static data visualization.

The major advantage of using QGIS is its accessible repositories offering support to design own application using different GIS features. In order to provide flexible and dynamic support in data visualization, we develop a standalone application which incorporates both static and dynamic

capabilities to analyze and display varied spatial patterns over maps using the captured data from various spatially enabled devices. The data is mostly associated with the location and therefore can be represented on the map. This approach will help in accumulating necessary data for various analysis and allow us to access the visualized data from anywhere in the world.

Currently, the system can illustrate a map of any region and top of that display an informative layer locating various car objects as Points including its all geographical positions, in addition with other cellular and car information present in the traffic data. The visual display automatically adjusts itself and will appear as long as data is continuously provided to the system.

The usefulness of this application is only limited by the data which is provided to it. Data gathered is dependent on the application used to gather data such as mobile device (Android/ios). That is why we need an efficient system at a place to fulfill considering varied future needs. Henceforth, the developed application of our system is not limited with traffic analysis, yet can be used to gather other data with user-specific GPS data vehicle tracking, etc. and allow performing the necessary analysis which will eventually help decision makers in making a smarter decision for future development.

Preface

This master thesis, “A Flexible Way to Dynamically Visualize Spatiotemporal Data in a Geographic Information System” was performed by Soma Das for the course TTM4905 - Communication Technology, Master’s Thesis. This project was carried out for 22 weeks from February 6 - July 3 in the spring semester of 2018 at the Intelligent Transportation Systems (ITS) lab as a part of my 2-years International Masters of Science study in Telematics-Communication Networks and Networked Services at the Department of Information Security and Communication Technology at Norwegian University of Science and Technology (NTNU), Trondheim.

This master thesis report is my original research work except the references and acknowledgments being used.

Soma Das

July 2018

Trondheim, Norway

Acknowledgement

I would like to thank my responsible professor, Prof. Peter Herrmann for introducing me to this project and continuously supervising my work. He has been very helpful in providing me guidance and right directions throughout the accomplishment of this project. He has always been very appreciative towards my achievements and progress in this project.

I would also like to thank my supervisor, Ph.D. student Ergys Puka and my co-supervisor, Thomas Levin from Statens vegvesen for support, Prof. Frank Alexander Kraemer for being my project chairperson during oral presentation of Communication Technology, Master's Thesis and, Prof. David F Palma for wonderfully giving me an overview for the structuring of thesis.

Laurent Paquereau, Mona Nordaune and Pål Sturla Sæther, the staff members at the Department of Information Security and Communication Technology (IIK) were very kind and cooperative in facilitating administrative guidance, providing workspace and trouble-free access in Intelligent Transportation System lab and, also allocating required technical resources.

A deep gratitude for my family to be always present showing unbreakable trust and faith in my capability. Special thanks to my friends who were encouraging me to work harder and constantly reminding to accomplish quality work.

Contents

List of Figures	xiii
List of Tables	xvii
List of Algorithms	xix
List of Acronyms	xxi
1 Introduction	1
1.1 Objectives	3
1.2 Methodology	3
1.3 Motivation	3
1.4 Publication	4
1.5 Statens vegvesen Collaboration	4
1.6 Thesis Contribution	4
1.7 Thesis Structure	4
2 Background and Literature Review	7
2.1 Terminologies	7
2.1.1 What is Spatiotemporal data?	7
2.1.2 What are Spatiotemporal Databases?	7
2.1.3 What is Geospatial Data?	8
2.1.4 What is Geovisualization?	10
2.2 Geographic Information System (GIS) Technologies	10
2.2.1 What is GIS?	10
2.2.2 GIS Tools for Geovisualization	10
2.3 Quantum- Geographic Information System (QGIS)	12
2.3.1 QGIS- Desktop Application	13
2.3.1.1 QGIS- Desktop GUI	13
2.3.2 QGIS- Browser Application	14
2.3.3 QGIS- Server Application	15
2.4 Background Work	15
2.5 Summary	16

3	Environment Setup for Dynamic Visualization System	17
3.1	System Blueprint for Flexible & Dynamic Visualization of Spatiotemporal Data	17
3.2	Setting up Development Tools for Building Python Standalone Application	19
3.2.1	Setting up Python 2.x	19
3.2.2	Setting up IDE or Editor	20
3.2.3	Setting up Qt/PyQt4	20
3.2.4	Setting up Linux	21
3.3	How to make Python and QGIS work together?	21
4	NTNU-QGIS Web Application & QGIS Server Setup	23
4.1	Apache server	23
4.2	Implementation of QGIS-NTNU Web-Application	24
4.2.1	NTNU-QGIS Web-Application Setup	24
4.2.1.1	Framework Architecture: MVC model	25
4.2.1.2	Login- Web Page	26
4.2.1.3	Uploading file to QGIS Database Web Page	26
4.2.1.4	Dashboard Web Page	27
4.3	QGIS Server	28
4.3.1	QGIS Server Setup	28
4.3.2	How Web-Application interacts with QGIS Server	29
4.4	Brief Introduction to NTNU-QGIS Web API	32
4.5	Summary	33
5	Spatial Database Setup	35
5.1	Why PostgreSQL?	36
5.2	PostgreSQL Setup	36
5.2.1	PostgreSQL Installation	36
5.2.2	Creating a Postgres Database User	36
5.2.3	Creating a new Database	37
5.2.4	Creating a Data Table	38
5.2.4.1	Making a table using Postgres-Bash Prompt	39
5.2.4.2	Alternate option to create table : Postgres GUI	40
5.3	PostGIS: Creating Spatiotemporal Databases	40
5.3.1	What is PostGIS?	41
5.3.2	Setting Up PostGIS	42
5.3.2.1	Installation of PostGIS	42
5.3.2.2	Implementation of PostGIS on ntnu-qgis Database	42
5.3.3	Testing PostGIS Functionality	42
6	Spatial Reference System in Visualization	47

6.1	What is Spatial Reference System (SRS) or Coordinate Reference Systems (CRS)?	47
6.1.1	Projected Coordinate Systems	49
6.1.2	Coordinate Reference System (CRS)- Formats	51
6.2	Implementation of Storing and Representation of Geographical Features in Database	52
6.2.1	Simple Feature Model for SQL	52
6.2.2	Adding spatial geometry field to table	55
6.3	Summary	58
7	Design Implementation of Dynamic Visualization of Spatiotemporal Data using PyQGIS	59
7.1	PyQGIS- Application Overview	59
7.2	Blueprint of Standalone Script using PyQGIS	60
7.2.1	Skeleton of PyQGIS Script	60
7.2.2	First GUI View of PyQGIS Application	61
7.3	Implementation of PyQGIS Application	62
7.3.1	Static Visualization & Analysis	62
7.3.1.1	Static Data Visualization Process	63
7.3.1.2	Analysis: Specific Patterns in Spatiotemporal Data	65
7.3.2	Dynamic Visualization & Analysis	67
7.3.2.1	Flow 1: Uploading & Storing Spatiotemporal Data to the Database	67
7.3.2.2	Flow 2: Dynamic Visualization of Data on Map Canvas	69
7.3.2.3	Stage 1: Creating WMS Layer (Basemap)	70
7.3.2.4	Stage 2: Creating Vector Layer from Database data	71
7.3.2.5	Stage 3: Checking for Data Updates	74
7.3.2.6	Flow 3: Analysis of the Data	74
7.3.2.7	Find out Position Coordinate of random Car Feature	77
7.3.2.8	Creating Connection with the QGIS Server	77
7.3.2.9	Downloading Map- Progress bar	78
7.3.2.10	Quit Button	78
8	Discussion and Conclusion	81
8.1	Discussion	81
8.2	Limitation and Challenges in Current Work	82
8.3	Conclusion	83
	References	85
	Appendices	
A	Source Code: Dynamic Visualization System	89

.1	Source Code: Static Visualization	89
.2	Source Code: Dynamic Visualization	91
.3	Source Code: PyQGIS-Application	93

List of Figures

1.1	This figure shows the various visualization patterns of spatial form of Road Traffic Data using GIS, the reference from [Hen, Gro, NEW]	2
2.1	This image shows a pictorial representation of Vector and Raster forms of data, adopted from [Uni]	9
2.2	This image shows various components forming a GIS System, adopted from <i>Source: https://www.cdc.gov/gis/</i>	11
2.3	The figure shows the available three types of QGIS Applications	12
2.4	The figures shows the five Components of QGIS-GUI Desktop Application	13
2.5	The QGIS-GUI Browser Application	14
3.1	Blueprint of Dynamic Visualization System Architecture	18
4.1	This image shows a basic Apache Web Server Architecture, adopted from <i>source:https://guidespratiques.traduc.org/guides/vo/solrhe/images/</i>	23
4.2	This image shows the basic interaction overview between a Web-client with a Web Server	24
4.3	This image shows an interaction diagram within MVC (Model View Controller) Model pattern, adopted from [ore]	25
4.4	The image shows the NTNU-QGIS web application- Login web page . .	26
4.5	The image shows the NTNU-QGIS web application- Uploading files web page	27
4.6	The image shows the NTNU-QGIS web application- Dashboard web page	27
4.7	This figure is the outline of NTNU-QGIS Web-Application interacting with QGIS Server [Liz]	30
4.8	This figure is the image of the QGIS Server response in the form of XML Document to GetCapabilities HTTP Request	31
4.9	This figure is the Static image of the QGIS Server response to GetMap HTTP Request	32
4.10	This image shows the central Web API Architecture holding all the business logic, adopted from [Mic]	33

5.1	A Basic Architecture of Database System	35
5.2	The Various PostgreSQL Permissions to Database User [Qgia]	37
5.3	The image shows the screenshot image of Verification of becoming Postgres User	38
5.4	Creation of ntnu-qgis Database	38
5.5	The view of route table schema	39
5.6	The view of pgAdmin GUI on Apache- Localhost	40
5.7	Configuration of PostGIS on ntnu-qgis Database	42
5.8	PostGIS Point function on ntnu-qgis Database	43
5.9	PostGIS Point function testing at Position 1,1	44
5.10	An example reference image showing how PostGIS- Point function is used to display location using data records. <i>source: mrbool.com/how-to-track-users-registration-location-in-g oogle-maps/33562</i>	45
6.1	This image shows a basic Coordinate System known as Cartesian system. <i>Source: https://svn.osgeo.org/qgis/docs/</i>	48
6.2	A CRS defined conversion of location on a 3-Dimensional plane to a 2-Dimensional plane. <i>Source: http://ayresriverblog.com</i>	48
6.3	The two types of CRS	49
6.4	A CRS defined three Coordinate locations (Oslo, Boulder and Mallorca) on a 2D Map. <i>Source: https://www.earthdatascience.org/</i>	50
6.5	A reference figure with CRS translation of 3-Dimensional Globe to a 2-Dimensional flat surface Map. <i>Source: CA Furuti, progonos.com/furuti</i>	51
6.6	A Simple Feature for SQL (SFS) Model defining various geo-spatial data from Point, Lines, and Polygons	53
6.7	A schema of Spatial Reference System Table	54
6.8	EPSG:4326 - the geographic lat/lon reference system using the WGS 84 ellipsoid	55
6.9	A SQL command to Alter "route" table by adding Geomtery field	56
6.10	Performing Point Constraint on table "route"	56
6.11	An addition of Geometry field with "Point" constraint on table "route"	56
6.12	A Geometry fields with Latitude and Longitude object values	57
6.13	A PostgreSQL GUI having Spatial object field "the_geom" in table "route"	57
7.1	This figure will be referenced as Python Standalone Application Overview, some images adopted from [ESR]	59
7.2	This figure shows Logic behind Flexible and Dynamic Visualization of Spatiotemporal-Data in PyQGIS Application	60
7.3	This figure shows PyQGIS- GUI View, the background Google map image adopted from <i>source: http://mirafra.com/tripsy/2214585-google-maps-wallpaper/</i>	61

7.4	This figure shows "About"-Menu Bar Option, providing description of the PyQGIS Application	62
7.5	This figure shows "File"-Menu Bar Option, providing access to local computer file system to upload Data file	63
7.6	This figure shows Flow Diagram for "Static Visualization and its Analysis"	63
7.7	This figure shows the Static Visualization of Spatiotemporal data stored in local CSV file	65
7.8	This figure shows "Select Vector Layer"- ToolBar Option, providing access to select any layer from all the layers created and make query	66
7.9	This figure shows the zoom out Map-View for Queried data (Signal Strength > 20)	66
7.10	This figure shows Flow Diagram for "Dynamics Visualization and its Analysis"	67
7.11	The flow 1: Uploading and Storing Spatiotemporal Data to the Database	68
7.12	This figure shows "Connect to NTNU-QGIS webpage"-ToolBar Icon, providing direct access to NTNU-QGIS login webpage to upload data .	68
7.13	This figure shows "Connect to PostgreSQL database"- ToolBar Icon, providing access to (ntnu-qgis) Database	69
7.14	The flow 2: Dynamic Visualization of spatiotemporal database data on Map-view process	69
7.15	The figure shows the WMS/WFS basic architecture for serving map over the cloud adopted from [fSHE]	70
7.16	This figure shows "Add WMS Map"-ToolBar Icon, to create and display Norge BaseMap Layer	71
7.17	This figure shows "Create vector layer"-Toolbar Option, creating vector layer from data stored in "ntnu-qgis" Database	72
7.18	This figure shows Dynamic Vector Layer Visualization on the Map Canvas	73
7.19	This figure shows the flow diagram for (database) Data Analysis using "Select vector Layer"-Toolbar Option	74
7.20	This figure shows the Analysis Result obtained using "Select Vector Layer"-Toolbar Option on Database Vector-Layer	75
7.21	This figure shows flow of analysis process directly on the database data	75
7.22	This figure shows "Make a deeper Analysis"- ToolBar Option, and steps to make query	76
7.23	This figure shows "Make a deeper Analysis"- ToolBar Option, Analysis result	76
7.24	This figure shows "Where AM I? "-Toolbar Option, to capture specific Position Coordinate(Longitude,Latitude) by click of a Mouse	77
7.25	This figure shows Map view capturing specific car feature Position Coordinates (Longitude,Latitude)	77

7.26	This figure shows "Connect to QGIS Server"- ToolBar Option, providing access to QGIS Server	78
7.27	This figure shows "Download Map"-Progress Bar Option, creating PDF of layers Displayed on Map-Canvas	78

List of Tables

6.1	A table contain various SRID	54
7.1	An Attribute table containing all Car Features saved in a CSV file . . .	64

List of Algorithms

7.1	Logic to create Vector Layer from CSV file	64
7.2	Requesting Norge WMS services and creating Basemap Raster Layer	72
7.3	Basic logic to create database Vector layer	73
7.4	Basic logic to print the Map Canvas in PDF	79

List of Acronyms

API Application Programming Interface.

CRS Coordinate Reference System.

CSS Cascading Style Sheets.

CSV Comma Separated Values.

DBMS Database-Management System.

EPSG European Petroleum Survey Group.

GCD Greatest Common Divisor.

GDAL Geospatial Data Abstraction Library.

GIS Geographic Information System.

GPL General Public License.

GPS Global Positioning System.

GUI Graphical User Interface.

HTML Hypertext Markup Language.

HTTP Hyper Text Transfer Protocol.

IDE Integrated Development Environment.

IGNF Institut Geographique National de France.

IIK Department of Information Security and Communication Technology.

ITS Intelligent Transportation Systems.

KML Keyhole Markup Language.

MVC Model View Controller.

NTNU Norwegian University of Science and Technology.

OGC Open Geospatial Consortium.

ORDBMS Object-Relational Database Management System.

PHP Hypertext Preprocessor.

QGIS Quantum-Geographic Information System.

RDBMS Relational Database Management System.

RTD Round Trip Delay.

SFS Simple Feature for SQL.

SQL Structured Query Language.

SRID Spatial Reference System Identifier.

SRS Spatial Reference System.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.

UTM Universal Transverse Mercator.

WCS Web Coverage Service Interface Standard.

WFS Web Feature Services.

WGS World Geodetic System.

WKB Well-Known Binary.

WKT Well-Known Text.

WMS Web Map Services.

XML Extensible Markup Language.

Chapter 1

Introduction

Before the birth of linguistics, there was drawing. Prehistoric sketches, paintings shared out history, plans, and ideas. *When something gets too difficult to understand, occasionally resorting to this uncomplicated and straightforward language of data visualization simplifies to a concept one can comprehend effortlessly.*

Utilizing Data visualization approach puts complex data into a pictorial or graphical format, allowing more ease in understanding the state of the system and identify hidden patterns to produce significant result for decision makers.

In the early phase of data visualization, one of the foremost ways existed to visualize data was using capabilities within Microsoft Excel. Starting from an easy-to-use spreadsheet and gradually painstakingly create a simple and comprehensive graphic to assist in conveying an idea or a message and providing a better understanding of a business trend. Excel was considered to be the best tool until data evolved to an extent that it was not possible for Excel to handle size and in addition to professionals growing demand for more complex visualization than a simple line, pie, or bar graphs. This is when the modern era of data visualization began [THE].

In recent years, curiosity in visualizing and analyzing the spatiotemporal¹ data has grown noticeably in the scientific research community. Researchers in the diverse field of studies such as Ecology, Environmental-health, Traffic Network Analysis, and Climatology are more and more encountering with the task of analyzing complex data that are *highly multivariate, georeferenced*², often represented as Maps, and *temporally correlated*, as in longitudinal/latitudinal or other some more series of time structures [BCG14].

This reflected the existence of well-formulated uncertainties in mind or hypothesis where location plays a paramount role, of spatiotemporal data with sufficient features and, of significant statistical methodology.

¹of or relating to both together spatial (space) and temporal (time) qualities

²geographically referenced, associated with locations in physical space

2 1. INTRODUCTION

All possible data visualization is interactive and its analysis can be a starting point for exploration to out of sight patterns which can be revealed and questions receive meaningful answered. It is very important as part of the process of understanding the results acquired in spatiotemporal data analysis, for interpreting the association between actual geography and how that real-world reality is captured in a software database converted into data matrix containing both locations specific as well as attribute data. With the increasing demands on complex data visualization, there is now widely available and easily accessible Geographic Information System (GIS) software which are continuously been used to handle spatiotemporal data. Furthermore, implementing a program for varied spatial data analysis is highly encouraged if supporting software is available [HH03].

Statistics governs to a great extent of what we do in reality. Since over the years, traffic data aggregation has been a serious and far-reaching factor of inaccuracy in nearly all road safety studies.

So far it has been estimated that approximately 40% of people every day on an average spend one hour on the roads [dvi]. Therefore, traffic data analysis is of high-priority. Using data visualization techniques helps to discover hidden valuable information present in the data providing aid in exploration and narrative after specific analysis. The figure Figure 1.1 below gives an illustration of analyzing various pattern present in road traffic data of spatial form facilitated by using GIS software.



Figure 1.1: This figure shows the various visualization patterns of spatial form of Road Traffic Data using GIS, the reference from [Hen, Gro, NEW]

1.1 Objectives

The main objectives of this thesis work are as follows:

- Create a server for storing and analyzing location-specific data and making predictions out of that.
- Allow the reception of data coming from various Android devices by the server.
- Integrate the server with QGIS³ such that the two systems can communicate in real-time.
- Display various geographical patterns using QGIS for the data gathered and stored in the database.

1.2 Methodology

To achieve the above-mentioned objectives, the agile methodology was used. In the start, we assemble theoretical knowledge and understanding of the various tools being used. The main source of gathering materials were Google, QGIS API Libraries, PyQGIS Programming Books and IEEE Xplore.

Based on the knowledge acquired, the tasks performed as:

- To allow the reception of data from various devices, a user interface is developed in the form of web application using HTML5/CSS, JavaScript and PHP.
- To store the data, the database is created using PostgreSQL. In addition, PostGIS functionality is implemented to convert the data into spatial form.
- A standalone python script is developed using QGIS libraries, PyQT and QT creator to integrate with the database to capture data in real-time.
- Data is displayed using QGIS modules and analyzed making specified SQL queries.

1.3 Motivation

This thesis is a research part to the previous project work performed by Prof. Peter Herrmann and Ph.D. Ergys Puka in the development of the Android application to collect cellular traffic road data attached with location information capturing together in the form of spatiotemporal data. Previously data was presented in a static way but this thesis inclination was to provide flexibility, scalability and dynamic approach

³open source free software used to view, edit and manipulate geographical data

to data visualization.

This work is vital in the area where complex data objects are to be visualized and analyzed to gather result for managing better road traffic. Performance is the main issue with this sort of complex information, so developing a system that is light-weight and powerful to handle a large amount of data and make a fast deeper analysis with dynamic support.

1.4 Publication

The previous work related to the literature part of this thesis has been published in the 2018, 10th International Conference on Communication Systems & Networks (COMSNETS)⁴ at Bengaluru, India under the title; *A way to measure and analyze cellular network connectivity on the Norwegian road system.*

1.5 Statens vegvesen Collaboration

The Norwegian Government Agency is popularly known as **Statens vegvesen**⁵ (English: *Norwegian Public Roads Administration*) which is responsible for the state including county public roads in Norway. The idea behind this thesis was first born after getting an introduction behind Norwegian road traffic data collection and its attributes, and its static visualization using the QGIS-Desktop platform.

1.6 Thesis Contribution

The details of the achieved contribution will be demonstrated in the later chapters of this thesis report. The main contribution can be summarized as:

A graphical user interface generated with python standalone application has been developed using various QGIS Libraries which incorporates both static and dynamic visualization capabilities. Along with that, a PostgreSQL database has been created with addition extending core capabilities using PostGIS to handle spatiotemporal data. The system is flexible and scalable to visualize any sort of raw data over the maps and gather useful graphical patterns during diverse analysis. In addition to this, a web application is designed in providing an ease to push data into the database from anywhere using any device.

1.7 Thesis Structure

This project thesis contains total of 8 chapters including this current introductory first chapter. A brief description of further chapters in this report are as follows:

⁴<https://www.comsnets.org>

⁵<https://www.vegvesen.no/>

- **Chapter 2: Background and Literature Review**
This chapter give an overview of the previous related work and describing various terminologies used in this project.
- **Chapter 3: Environment Setup for Dynamic Visualization System**
This chapter provides a complete installation process to setup environment needed to build Dynamic Visualization system.
- **Chapter 4: NTNU-QGIS Web Application & QGIS Server Setup**
This chapter provide the implementation of project web application, Apache web server and QGIS web server and, demonstrating there purposes for this project.
- **Chapter 5: Spatial Database Setup**
This chapter demonstrate the complete process in creating PostgreSQL database with implementing PostGIS functionality.
- **Chapter 6: Spatial Reference System in Visualization**
This chapter provide a complete insight view over spatial objects creation and there formation over geographic maps.
- **Chapter 7: Design Implementation of Dynamic Visualization of Spatiotemporal Data using PyQGIS**
This chapter describes the implementation process in developing PyQGIS application.
- **Chapter 8: Discussion and Conclusion**
This chapter summarizes the complete project thesis along with challenges and limitation occur in the process.

Chapter 2

Background and Literature Review

This chapter will provide a brief introduction of the previous work which brought the idea behind this thesis. This chapter also provides a general introduction to briefly understand about terms and technologies used to accomplish the goals.

2.1 Terminologies

2.1.1 What is Spatiotemporal data?

To understand this word **Spatiotemporal** lets break down it into two parts as "Spatial" and "Temporal". For the term **Temporal**, its characterization appears when a series of images are taken at a distinctive time. The interrelationship between the captured images is generally used to monitor the dynamic changes of the object within the course of time. The **Spatial** characterization come to pass when image is analyzed.

Spatial and Temporal features form a significant portion in the vast amount of data produced by various mobile devices, Geographic Information System (GIS) systems, data monitoring applications and many other processes. The "spatial" is associated with direction, shape, direction, etc. and "temporal" is related to duration, occurrence time, etc. These parameters have to be extracted from the raw form of data thereby useful conclusions can be drawn [Quo].

2.1.2 What are Spatiotemporal Databases?

A Spatiotemporal Database is a database that is responsible for dealing with data which encloses both space and time information. The word "deal" here implies to the managing, creation, editing and/or updating of data. The most common examples include:

- Historical tracking of activities of the Tectonic Plates on the Earth's surface.

8 2. BACKGROUND AND LITERATURE REVIEW

- An index of species in a particular region, where with time new species are introduced or existing species are extinct or migrated.
- A wireless communication network database, which may occur for a short time specifically to one particular geographic region.
- Tracking of moving objects e.g. car, planes, etc., which in most cases occupy one single position at a given time.

Spatiotemporal Databases are basically an extension of Spatial Databases. A spatiotemporal database encompasses spatial, temporal, and spatiotemporal database concepts, and capturing complex spatial and temporal aspects of data from raw information and deals with:

- Location of moving objects over unvarying geometry known differently as real-time tracking systems or moving objects databases [GS05] and/or
- dealing with gradually changeable geometries over time.

However there exist various relational databases¹ providing spatial extensions, but for practical reasons the spatiotemporal databases are not based on relational models. Primarily, because the data is multi-dimensional, and capturing complex structures and behaviors. By 2008, there was no RDBMS² product with spatiotemporal functionality but now the PostgreSQL³ a powerful ORDBMS product and the first among various popular Database-Management System (DBMS) to provide spatiotemporal functionality. PostGIS⁴ is a spatial database extender which extends the core capabilities of PostgreSQL. This extender provides support for geographical objects which allow location based queries to be run in SQL [en].

2.1.3 What is Geospatial Data?

Geospatial data as sometime known as Spatial data, is an information which has geographical aspect to it. In other words, it is a subset of spatial data which is simply a data which contains records indicating location of objects within a given coordinate system and/or city, an address, postal code or zip code included with it. The most evident example is a road map. What we see is the rendered result, but the features on the map are stored containing this type of information [Mat].

¹<https://www.sisense.com/glossary/relational-database/>

²<https://database.guide/what-is-an-rdbms/>

³<https://www.postgresql.org/about/>

⁴<https://postgis.net>

Geospatial data comes in many forms (shapes) and formats, and its structure is highly complicated to understand than simple tabular and/or even non-geographical geometric data. It is different because here data refers to as objects which contain complex figures like Points, Lines, Polygons and many other forms of shapes including scale parameters and elevation etc. located in geographic space [stu].

There exists two fundamental types or forms of geospatial data for computer storage and application and, for its representation as shown in Figure 2.1 are:

- **Vector Data:** This type of data make use of Point, Lines, Polygons to create and represent spatial features such as roads, cities, positioning moving objects like car, etc along with using other attributes on the map.
- **Raster Data:** On the other hand, this type uses cells to represent spatial features. In computer these cells are dots or pixels. Cities are single cells, road as linear sequence of cells. The most common example of raster form are Satellite images.

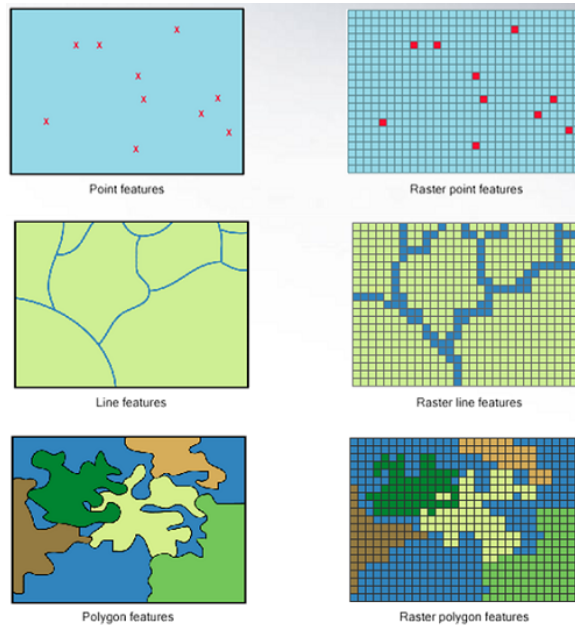


Figure 2.1: This image shows a pictorial representation of Vector and Raster forms of data, adopted from [Uni]

Both form of data are processed by using different methods for its representation. The raster is faster but vector is a corrector. How it is implemented is demonstrated in Chapter 7.

2.1.4 What is Geovisualization?

Geographic Visualization in short Geovisualization. It is a process that modifies and display complex geographical information (geospatial data) to make it in human understandable presentation.

For example, we consider a table containing Trondheim’s temperature information for the month of June. Would the table be more capable of providing as much insight like a visual image as a map of the same information? Probably not! Besides to the values, the map will also show the various patterns of the temperatures. This is one of the basic examples of the Geovisualization process. This complex raw information becomes knowledge by converting location-dependent information to visual patterns. In this thesis, we are using cellular traffic data collected from tracking moving car on Norwegian roads and analyzing various spatial patterns present inside the data [stu].

2.2 Geographic Information System (GIS) Technologies

2.2.1 What is GIS?

GIS stands for **Geographic Information System**. A simple interpretation implies *as a computer system which is capable to store and use data for describing places on the Earth’s surface*. However, GIS encloses more than just a simple computer system.

A GIS is a **framework** which is designed for capturing, storing, managing, displaying and analyzing data. As GIS integrates many types/forms of **data**, it can **analyze** spatial location and organizes layers of information (layer can be thought of card in a deck) into visualizations using various maps and/or 3D scenes. With this unique capability, the strength of GIS lies in its ability to reveals deeper insights into data, such as relationships, patterns between layers. Furthermore, **situations—helping people** make smarter decisions by using GIS to answer particular data-related questions [ESR].

The Figure 2.2 shows the various components in GIS as learned above.

2.2.2 GIS Tools for Geovisualization

Nowadays, thousands of organizations in practically every field are using GIS to develop maps that communicate, perform analysis and solve complex problems around the globe. GIS helps individual and organizations to better understand complex spatial structures and patterns as is has got great capability for visualizing many different kinds of data on one single map [Geoa].



Figure 2.2: This image shows various components forming a GIS System, adopted from *Source: <https://www.cdc.gov/gis/>*

There are many GIS tools available today for creating and analyzing Geovisualization images. Some of the most popular ones include [Glo]:

- **ArcGIS:** This is platform-based commercial tool developed by ESRI⁵ in 1999 and written in C++ . It dominates in the GIS area because it is simple to use, cloud functionality and providing exceptional maps.
- **QGIS:** A free open source cross-platform tool⁶ developed by Quantum-Geographic Information System (QGIS) Development Team in July, 2002 and, written in C++, Python and Qt. It allows viewing, editing/updating, and analysis of geospatial data.
- **Mapbox:** Is is an open source mapping platform-based tool⁷ founded in 2010 for custom designing both web and mobile maps.
- **Carto:** This is Service cloud computing platform-based tool⁸ released on September, 2011 and written in Ruby, JavaScript that offers drag-and-drop features capabilities for quickly creating both simple and sophisticated visualizations.

⁵<https://www.esri.com/en-us/home>

⁶<https://qgis.org/en/site/about/index.html>

⁷<https://www.mapbox.com/about/>

⁸<https://carto.com>

2.3 Quantum- Geographic Information System (QGIS)

Quantum-Geographic Information System (QGIS), also sometimes known as Quantum-GIS. It is a *free available and open-source cross-platform GIS application*. QGIS was developed by *QGIS Development Team* in July 2002 and released under License GNU General Public License (GPL) v2 [gnu]. The development of QGIS under this license means that it can be freely inspected and modify the source code to perform different or more specified tasks [ow].

Currently, QGIS is available to run on multiple operating systems including Mac OS X, most Unix platforms, and Microsoft Windows. It is written in C++ and development made extensive use of the Qt toolkit⁹. This makes QGIS look well structured, and along with that providing pleasing, easy-to-use Graphical User Interface (GUI). In addition to Qt, it integrates with various other open-source GIS packages. Mainly including GDAL¹⁰, GRASS GIS, SQLite, PostgreSQL/PostGIS and MapServer which provide access to the additional data formats. Plugins and Standalone Application are written in C++ and also Python which extends QGIS capabilities. Many plugins are available and can also be extended to dynamically add new functionality.

There are three types of QGIS applications in use today but the a mobile version of QGIS in under development process for Android since 2014:



Figure 2.3: The figure shows the available three types of QGIS Applications

⁹<https://www.qt.io>

¹⁰<http://www.gdal.org>

2.3.1 QGIS- Desktop Application

QGIS aims to be more advanced, comprehensive and a user-friendly desktop GIS application in open source world, providing some common functions and features. However, the initial goal of the project in the early development phase was to provide a GIS data viewer. But QGIS has reached to the point in its evolution today where it is being used by many organizations for their daily Geospatial data-viewing needs [JOU].

QGIS- Desktop allows users to create, edit, view and analyze the geospatial data, in addition to just creating and exporting geographical maps. It provides support for both raster and vector formats of data, including spatially enabled tables in PostgreSQL using its spatial extension PostGIS.

The GIS vector data is either stored as Point, Lines, or Polygons features and its format is Shapefiles with an extension .shp. While, raster forms of data are geo-referenced and multiple formats are supported including PNG, GEOTIFF, TIFF, etc. [Sou] as seen in Figure 2.1. In addition, it offer support for generates layer view using other files with CSV, .xlsx, XML formats and also support the use of data from external Web services sources, like Web Map Services (WMS) and Web Feature Services (WFS).

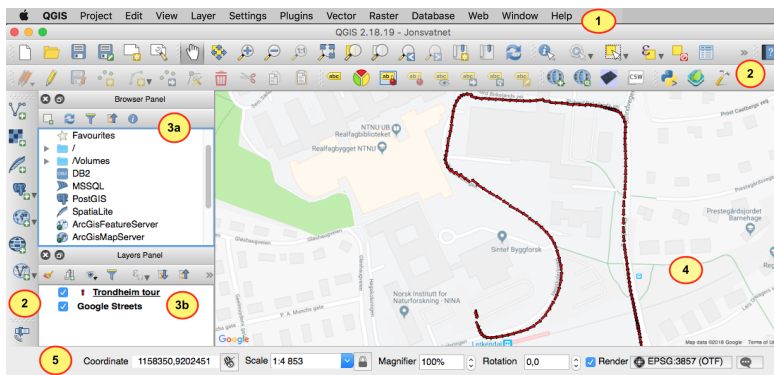


Figure 2.4: The figures shows the five Components of QGIS-GUI Desktop Application

2.3.1.1 QGIS- Desktop GUI

When QGIS-Desktop application starts, it appears with the GUI presentation as shown in Figure 2.4. The look of the GUI differs depending on the operating system and window manager [opc].

The Desktop-GUI is divided into five different functional components as marked in red circle on the above figure:

1. Menu Bar
2. Toolbars (top/left vertical corner)
3. Panels
 - a) Top- Browser Panel
 - b) Bottom- Layers Panel
4. Map View (Map Canvas)
5. Status Bar

The Menu bar and Toolbars provide various GIS functionalities for creating different patterns from the spatial data. The Top- Browser panel is the shortcut to navigate in the local platform file system while the Bottom- Layer panel is responsible for listing all the raster or vector data layers created. The Map view is like a canvas where these generated layers are displayed on the map. The last, Status panel gives details about the position coordinated of the spatial objects as well the projection used to display them accurately on the 2D globe. This is further described in Section 6.1.1.

2.3.2 QGIS- Browser Application

The QGIS- Browser application provides easy navigation to the filesystem in your computer and databases for managing geospatial data as shown in its GUI representation Figure 2.5. A quick access to vector data files (e.g. MapInfo files or Shapfiles), databases (e.g. SpatiaLite, PostgreSQL/PostGIS, or Oracle) and for raster file making WMS/WFS¹¹ connections [opb]. The shortcut to it is also available in Desktop Application.

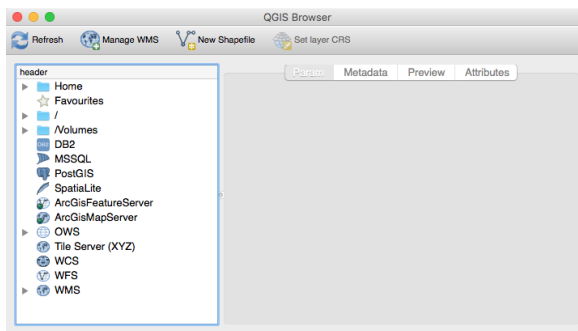


Figure 2.5: The QGIS-GUI Browser Application

¹¹<http://www.opengeospatial.org/standards/wms>

2.3.3 QGIS- Server Application

QGIS Server Application is an *open source WMS 1.3, WFS 1.0.0 and WCS 1 1.1.1 implementation* which, in addition, implements most advanced *cartographic features for thematic mapping* [opa].

The QGIS Server usually run as a *FastCGI/CGI (Common Gateway Interface) module*, coded in C++ within a web server (e.g., Apache, Lighttpd). It uses QGIS-Desktop as a backend for its GIS logic, meaning both the applications uses same visualization Libraries for map rendering. This makes the appearance of the maps on QGIS- Desktop look alike as published on the web using QGIS Server.

QGIS server provides support for Python plugin facilitating fast and efficient development and deployment of new features and also integrating with existing features. The detailed description about the use and purpose of QGIS Server, along with its installation for this project design is provided in Chapter 4 Section 4.3.

2.4 Background Work

The information we are using to work with in this project consists of two elements, specifically spatial features and attributes data. As previously described about geospatial (spatial data) in Section 2.1.3, the examples of spatial features which you might find on a map including city, zip code, etc. In this case, it is data from moving car object whose changing position over time is its spatial feature. On the other hand, "attribute data" provides detail description about the characteristics of the spatial features which may be discrete or continuously value and/or quantitative or qualitative.

The previous work contribution to this project is to provide data. Here, it is cellular car traffic data gathered using Android application developed by my responsible professor Prof. Peter Herrmann along with my supervisor Ph.D Ergys Puka.

The Android device when placed inside the car, it starts gathering the details about the car as it moves forward. After a certain time-stamp usually taken after a second interval, it updates and stores the new information by sending to a remote server. The data being collected was later fetched into a form of a CSV file containing all records. These records are basically the attributes about each car feature changing over time. A car is treated as a spatial object here because it carries geographical aspect which is included with the other cellular information collected. The important thing to notice here, the geographical aspect imply here only to the *geographic coordinates* of the car positioning at a specific time mainly as a combination of *Latitude, Longitude, and Altitude*, and the other attribute data is the cellular information (*Communication Protocol, Signal Strength*), Server connection information (*Round trip delay*), along

with other categories including *Speed, Bearing, Accuracy and, Total number of Satellites and Satellites in fix.*

The Android application collects information and sends to the server where server collects data using either UDP or TCP connection. The server is implemented using Reactive blocks¹² which is an Eclipse IDE plugin. The time in which data send to the server and acknowledged is recorded as "Round trip delay" as the record in the attribute data.

2.5 Summary

When working with a spatiotemporal form of a data normally we are interested in the properties of that data which can make the interpretation of data simple, easy and intuitive. This data contains information and patterns of complex form so it requires manipulation and visualization techniques for human understanding for extracting useful information and making smarter decisions [Quo].

As acquired knowledge from above descriptions, QGIS is available in three different types of GIS functional applications, but considering the objective of this project to provide flexibility and dynamism for which desktop version has some limitation to work with. It only provides a **Static view** of the spatial data. With the word "static" here implies to images presented on the Map canvas (Map view) is constant. As QGIS-Desktop stores data in the form of table temporary into its own database and during visualization process once the image is displayed on the map canvas and simultaneously the same data is updated it makes no difference on the map view. We have to restart the same process of upload the entire data and make a new layer. The other most important drawback of using desktop version was slow performance according to the size of data. The computer system has to devote its maximum core processor to bring efficiency and increase its performance. This makes it highly unsuitable to run other application simultaneously with QGIS Desktop.

In order to overcome this limitation we designed separate Python standalone application which is light weight and easily adoptable to run in other operating system using various QGIS Libraries. The system design is more salable and flexible to handle spatiotemporal data which will be describe in later chapters. The next Chapter 3 describes the environment setup for system designing.

¹²<http://reference.bitreactive.com>

Chapter 3

Environment Setup for Dynamic Visualization System

The system architecture is designed to bring scalability and flexibility in real-time geospatial data visualization. The main aim is to capture meaningful results according to specific requirements for making smarter decision from the data collected. The system architecture consists of four components listed below and this chapter demonstrates the process to setup tools for building Python Standalone application.

1. Android Application
2. NTNU-QGIS Web- Application
3. Spatial Database
4. Python Standalone Application

3.1 System Blueprint for Flexible & Dynamic Visualization of Spatiotemporal Data

The designing of system architecture is shown in Figure 3.1. The flow of the system is shown by 8 yellow markings and each marking performs some function to create dynamic visualization of data collected and for further deeper analysis. An overview of the working of the system:

- **In marking 1:** Firstly, the data is gathered by using Android application which was described in Section 2.4. Its not a restriction the data can be collected from any device or application but in this work data from Android app is considered.
- **In marking 2:** The next step is the second main component of the system, a Web Application named "NTNU-QGIS" is created which is basically a responsive web interface to upload the data of any format from anywhere. This application is compatible to run in any operating system. Its implementation is detailed in the following Chapter 4 under Section 4.2.

- **In marking 3:** The data which is uploaded by user using a web-application is stored directly in PostgreSQL database. The database created is named "ntnu-qgis". It is the third important component in the system architecture and its complete installation and working process is described in Chapter 5 and Chapter 6.
- **In marking 4:** It is the backbone of the whole system, development of **Python Standalone Application**. It makes the connection with the database to receive continuous data and also the web-application API is embedded. The implementation of this application is elaborated in Chapter 7.

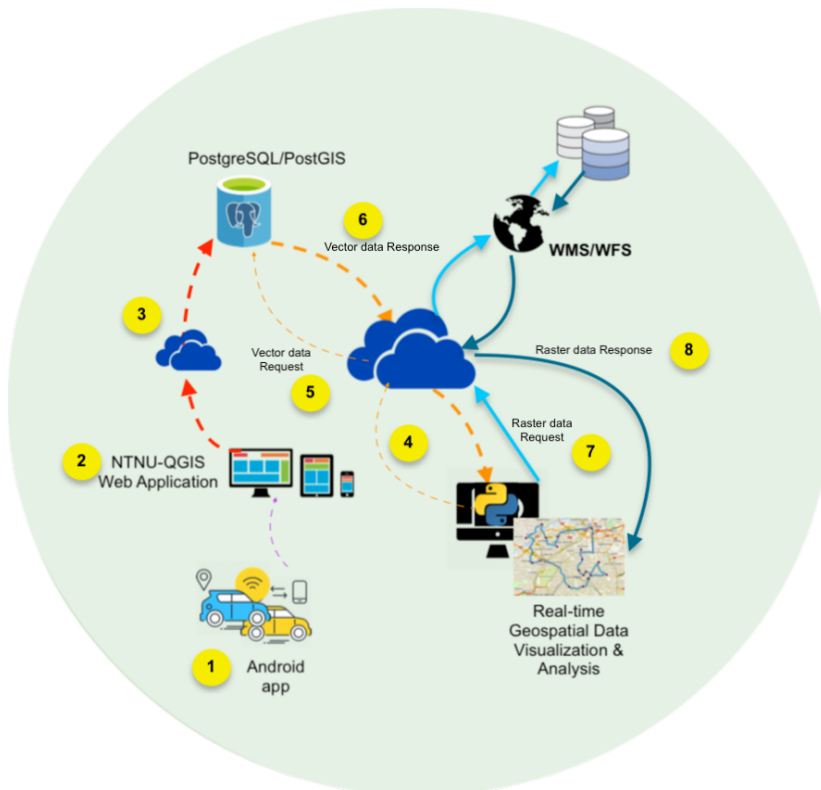


Figure 3.1: Blueprint of Dynamic Visualization System Architecture

- **In marking 5 and marking 6:** The Python Standalone Application requests the database for vector data and to form vector layer for visualization respectively.
- **In marking 7 and marking 8:** The application requests for web services for providing with raster data and to create geographic base map view to place

the data at correct locations respectively.

- The further involves various analysis processes by making a query on the data which is demonstrated in Section 7.3.2.6.

3.2 Setting up Development Tools for Building Python Standalone Application

The **Python Standalone Application** is the backbone of the entire system and in GIS terms is called a "PyQGIS" script. The standalone application means a separate computer process which is not a part of any packed together software. This PyQGIS script is a portable interactive GUI application written in python and uses various different QGIS modules for producing GIS functionalities. That is why it is named as PyQGIS.

In order to develop a robust PyQGIS script, a proper development environment is a must. The next following sections describes the tools used for building this GUI-standalone application for dynamic visualization of spatial data.

3.2.1 Setting up Python 2.x

Depending on which operating system is used, python might be pre-installed. The main key point to be noted here is that **QGIS is only compatible working with Python version number 2.x**. The installation of Python 2.7 version on Ubuntu has been executed using following steps [Teca]:

- **Step 1: Prerequisites Installation**

```
$ sudo apt-get update
$ sudo apt-get install build-essential checkinstall
$ sudo apt-get install libreadline-gplv2-dev libncursesw5-dev
libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2
-dev
```

- **Step 2: Download Python 2.7.14**

```
$ cd /usr/src
$ sudo wget https://www.python.org/ftp/python/2.7.14/
Python-2.7.14.tgz
```

Extract the downloaded package:

```
$ sudo tar xzf Python-2.7.14.tgz
```


– **Step 3: Commands to Compile Python Source code**

```
$ cd Python-2.7.14
$ sudo ./configure --enable-optimizations
$ sudo make altinstall
```

– **Step 4: Check the Python version before using QGIS**

```
$ python2.7 -V
```

3.2.2 Setting up IDE or Editor

This is a matter of personal choice of using either IDE or an editor. For this project, the best suitable was "PyDev". It is a free available IDE for the development of python scripts. Benefit in using IDE as it helps in providing an ease to navigate through the code, suggestion to errors and display classes, methods, and attributes which makes the process easier during debugging. These two commands used to install PyDev:

```
$ sudo apt-get install eclipse
$ sudo apt-get install eclipse-pydev
```

3.2.3 Setting up Qt/PyQt4

PyQt is one of the favoured cross-platform Python API interface to QT¹, the C++ framework on which QGIS is build on. To produce some output GUI from our PyQGIS script, we use PyQt to provide all the GUI elements as QGIS is based on this QT framework.

There are different installation process for different operating systems. For Linux, use package manager to install Qt designer and PyQt. It is important to install both for the development of interactive user interface and for this script we installed PyQt version 4. The setup commands as follows:

```
$ sudo apt-get update
$ sudo apt-get install python-qt4
$ sudo apt-get install libxml2-dev libxslt1-dev python-dev
$ sudo apt-get install python-lxml
$ sudo apt-get install xvfb
```

¹is the GUI application development framework owned by Nokia

3.2.4 Setting up Linux

The entire system is implemented on Linux Operating system. The instability occurring in executing various QGIS Libraries on Windows and Mac OS was crashing the script after few seconds of start which in a way affecting the whole system. This was bringing stability and performance issues in visualization. Installation and implementation of other tools were quick and robust after testing on Linux operating system but stability was gained using only previous version of latest tools. This may be due to a reason that QGIS is not flexible enough to work with new environments.

3.3 How to make Python and QGIS work together?

The QGIS is written in C++ and it encompass nearly over 4000 core classes that together make up this GIS application. And among these around 75 % are Python enabled by the use of SIP². SIP tool helps to ease the process of creating Python binding for C and C++ libraries. *The "sip" files being compiled as a part of QGIS to support and provide the Python interface to the C++ coded core classes* [She14]. The main QGIS modules³ used for this application are:

- **qgis.core** (Core classes to use QGIS functionality in script)
- **qgis.gui** (Graphical User Interface classes for providing QGIS Map view)
- **qgis.networkanalysis** (classes for network analysis)
- **qgis.analysis** (classes for analysis related purposes)
- **qgis.utils** (classes for interacting with Map Layers)

²<https://www.3cx.com/pbx/sip/>

³<https://qgis.org/api/2.0/>

Chapter 4

NTNU-QGIS Web Application & QGIS Server Setup

This chapter is about the implementation of NTNU-QGIS web page and describes its purpose in this project thesis . In addition, the brief introduction to QGIS Server and its installation process along with its usage.

4.1 Apache server

The Apache, formally known as Apache HTTP Server, is a well-known *free available, open-source cross-platform web server*. It was designed 23 years ago by Robert McCool and initially released in 1995 under terms of Apache License 2.0 and, maintained by Apache Software Foundation. The basic architecture of Apache web server is shown in Figure 4.1 and the installation has been implemented referring to its online resource <https://httpd.apache.org>. We will be using Apache as our web server for this project work. As Apache is an establish and popular tool and commonly used as a web server for the various academic project which makes its resources availability easy.

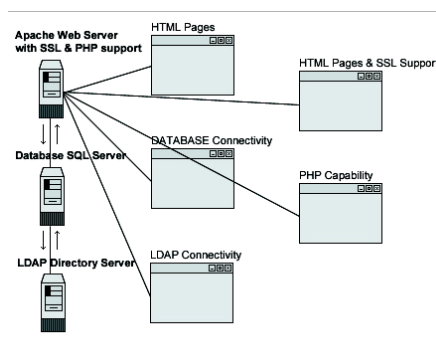


Figure 4.1: This image shows a basic Apache Web Server Architecture, adopted from [source:https://guidespratiques.traduc.org/guides/vo/solrhe/images/](https://guidespratiques.traduc.org/guides/vo/solrhe/images/)

When a web client (e.g. browser) makes an HTTP request for any webpage or content from the Web Server, the Apache will receive this request and interprets the content. By the word "interpret" here implies to determining whether the requested resource is of static type e.g. a basic webpage stored in an HTTP file format, or an application for example in the form of PHP file. The Apache will then determine the location of the content based on the path specified in the HTTP Request and give back a response to the browser as seen in Figure 4.2 [web].

The capabilities of web server are limited as it only waits for requests to arrive and later execute those requests. The web server does not allow web-client to directly interact with the application server and/or database and also with other resources. It is responsible to serve only specific information as specified in the web-client request.

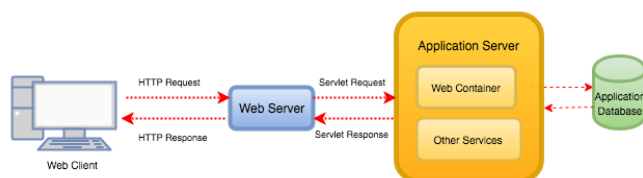


Figure 4.2: This image shows the basic interaction overview between a Web-client with a Web Server

4.2 Implementation of QGIS-NTNU Web-Application

The basic idea behind creating this "NTNU-QGIS" Web Application is to provide a graphical interface where with an ease the spatiotemporal data gathered by various devices can use it to push the data directly to stored into a PostgreSQL/PostGIS database created (refer Section 5.2.3). The further sections give a detail description about this web application.

4.2.1 NTNU-QGIS Web-Application Setup

Instead of starting developing the web application from scratch, an open source framework, CodeIgniter¹ is used which provides an environment including the boilerplate code for making an interactive web application. The framework adopts a Model View Controller (MVC) architecture which is described below Section 4.2.1.1.

¹<https://codeigniter.com>

4.2.1.1 Framework Architecture: MVC model

Model View Controller (MVC) is a software architecture pattern, commonly used for implementing user interfaces, this makes it a popular choice for architecting web applications in major programming languages.

In general, it divides an application logic into three separate yet interconnecting parts, contributing to modularity and an ease of collaboration and reuse. Moreover, it makes applications more flexible and acceptable to iterations. A Figure 4.3 illustrates an interaction diagram within MVC pattern. The most popular programming languages which have MVC frameworks that are used for developing web applications include C#, Java, PHP and we used PHP for this web application implementation [ew].

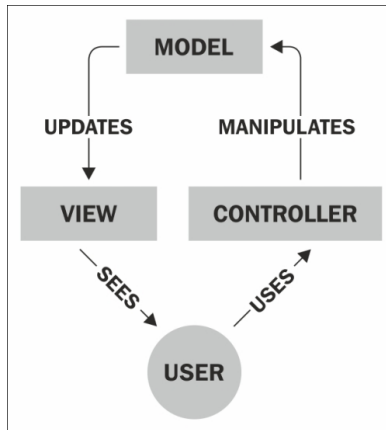


Figure 4.3: This image shows an interaction diagram within MVC (Model View Controller) Model pattern, adopted from [ore]

The three components of a MVC Model are:

- The **Model**: It is the central component in the Model. It manages the logic, data and controls the operation of an application.
- A **View**: It is the representation of the data output which can be a image, diagram or a chart.
- The **Controller**: This part is responsible for receiving the input and transforming it into set of commands for the Model and/or a View.

The interaction between the three components of a MVC Model:

- The **Model**: It is basically accountable for managing the application data. It receives the input from the Controller which was given by the user.
- A **View**: It is only responsible for specific format presentation of the Model.

- The **Controller**: it receives and responds to the user input, and also validates it and then forwards the input to perform interaction with the Model.

The designing of this web application, NTNU-QGIS consists of collection of three different web pages which are interlinked following MVC pattern. The three pages include:

- Login Web Page
- Uploading file to QGIS Database Web Page
- Dashboard Web Page (Data loaded Preview Page)

4.2.1.2 Login- Web Page

The Log-in page provides access to only authorized user at the moment. New users cannot sign-up. This is a temporary functional implementation for security reasons in this project but this is not the limitation. Using the specified email and password, a authorized user can log-in to the NTNU-QGIS web application. The view of the log-in page is shown below in Figure 4.4.

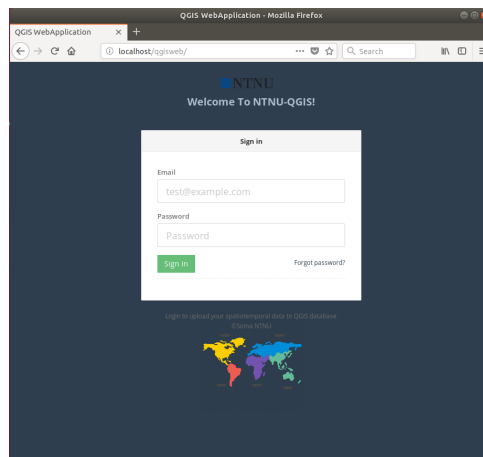


Figure 4.4: The image shows the NTNU-QGIS web application- Login web page

4.2.1.3 Uploading file to QGIS Database Web Page

After the successful log-in comes the next page, Uploading file web page as shown in Figure 4.5. The Upload interface where a user can upload spatiotemporal data in any file format into "ntnu-qgis" database (refer Section 5.2.3). The page is user-friendly, just by the mouse click on the "Browse" button will open the platform-dependent Windows Explorer to easily navigate through the local files system.

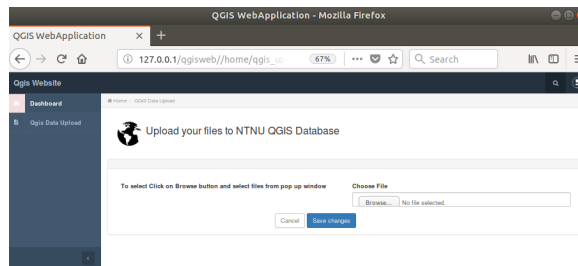


Figure 4.5: The image shows the NTNU-QGIS web application- Uploading files web page

4.2.1.4 Dashboard Web Page

The Dashboard web page shows the preview of the data uploaded by the user as shown in Figure 4.6. Here without externally connecting separately to the database, we can see and go through the uploaded data which is available inside the database at that time. When the new file or old file is updated this page automatically refreshes and display preview again of the current present data in the database. This play a good role during analysis process which will be seen later in Chapter 7 under Section 7.3.2.6.

id	communication_protocol	round_trip_delay	signal_strength	latitude	longitude	altitude	bearing	speed
1	HSLPA	1854	15	63.41856451	10.39061234	73	234.1	1.74
2	HSLPA	914	16	63.4185655	10.39067537	73	233.6	2.33
3	HSPA+	128	15	63.41854083	10.39052209	73	235.3	3.31
4	HSPA+	103	15	63.41852721	10.39044839	73	238.2	3.8009996
5	HSPA+	117	16	63.4185133	10.39050348	73	242.4	4.3109987
6	HSPA+	98	16	63.41850789	10.39027738	72	251.1	4.35
7	HSPA+	105	15	63.41850511	10.39016032	72	257.3	4.49
8	HSPA+	100	16	63.41851011	10.39044736	72	262.2	4.5409987
9	HSPA+	85	17	63.41851066	10.39044732	72	263.2	4.12
10	HSPA+	119	17	63.41850466	10.39087808	71	264.5	2.54

Figure 4.6: The image shows the NTNU-QGIS web application- Dashboard web page

As we have learned from the previous Chapter 2 that the QGIS application is available in three various forms. The two forms which are Desktop and Browser we have already discussed in detail (refer Section 2.3). A short description about the third form that is QGIS server was done in Section 2.3.3. Now let us see in detail about its implementation process along with its purpose.

4.3 QGIS Server

QGIS Server is an open source WMS 1.3, WFS 1.0.0 and WCS 1 1.1.1 implementation that, in addition, implements advanced cartographic features for thematic mapping. The QGIS Server is a FastCGI/CGI (Common Gateway Interface) application which is written in C++ and works together with a web server (e.g. Apache) [sitb].

QGIS-Desktop is used as a back-end for the GIS logic and for map representation by QGIS Server this makes the map visualization on both the application look similar during presentation.

4.3.1 QGIS Server Setup

QGIS Server allows the created project from the Desktop version to be published and can be visualized on browser using the Apache Server. In addition, it also allows to configure the properties of Geographic Web services e.g. WMS/WFS. Installation has been done only using the packages provided by QGIS development organization [sitc, os] and the steps followed includes:

- First, adding the **Debian QGIS repository**:

```
$ cat /etc/apt/sources.list.d/debian-gis.list
deb http://qgis.org/debian trusty main
deb-src http://qgis.org/debian trusty main

$ # Add keys
$ sudo gpg --recv-key DD45F6C3
$ sudo gpg --export --armor DD45F6C3 | sudo apt-key add -

$ # Update package list
$ sudo apt-get update
$ sudo apt-get dist-upgrade
```

- **Installing QGIS Server**

After the successful installation of the packages, now installing QGIS Server which should be used along with X server and without any interference with the QGIS Desktop with both installed on the same platform. Both the application work independently except using some common visualization modules.

```
$ sudo apt-get install qgis-server python-qgis
```

- **QGIS Server- Executable**

The QGIS Server executable defines the set of commands needed for its operation and, as it is a *FastCGI/CGI (Common Gateway Interface)* application, the

QGIS Server executable is *qgis_mapserv.fcgi*. The installed location usually is */usr/lib/cgi-bin/qgis_mapserv.fcgi* but could be different purely based on which operating system used.

– Create Apache Server Virtual Host

In order to access the QGIS server on the web browser, we require a web server and we are using Apache as a web server here as described in Section 4.1.

We created "qgis.ntnu" as a virtual host.

– Testing the QGIS Server by making a HTTP Request

By writing the following command and getting this output below shows that HTTP request for accessing QGIS Server works correctly. As we have not asked for any map web services to create geographical map therefore, the output shows **Service unknown or unsupported**.

```
$ curl http://qgis.ntnu/cgi-bin/qgis_mapserv.fcgi
```

Output:

```
<ServiceExceptionReport version="1.3.0"
xmlns="http://www.opengis.net/ogc">
<ServiceException code="Service configuration error">
Service unknown or unsupported</ServiceException>
</ServiceExceptionReport>
```

In the later section we will see the usage of this QGIS Server application.

4.3.2 How Web-Application interacts with QGIS Server

The QGIS Server uses the same visualization modules as used by the QGIS Desktop. So the images produced by both are similar. From the Figure 4.7 below we can have an idea about the connection process between NTNU-QGIS web application and QGIS Server.

Due to the time constraint there was not enough time to incorporate this additional feature on the NTNU-QGIS web-application but it is tested in other ways to produce the similar result. When the HTTP request is made from the internet browser it is send to the QGIS Server to access it. The QGIS Server request the various Web Map server for the geographical map and later display the GIS features on the browser containing the WMS services along with the data.

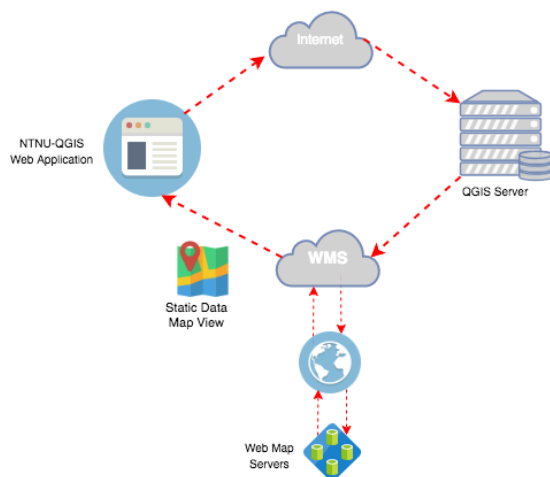


Figure 4.7: This figure is the outline of NTNU-QGIS Web-Application interacting with QGIS Server [Liz]

A Web Map Service (WMS) is a standard protocol developed by the Open Geospatial Consortium (OGC) in 1999 for serving georeferenced map images over the Internet. These images are typically produced by a map server from data provided by a GIS database. - Wikipedia

The map requests are made in two forms:

– **WMS GetCapabilities Request**

In this request, by pointing to any WMS web-client to the GetCapabilities URL, the response receive is in the form of a XML document stating Web Map Server’s Metadata as shown in Figure 4.8. In the Metadata information it includes, the number of layers being served, the geographical coverage area, the WMS version, etc. The Request is placed in this format:

```

http://qgis.ntnu/cgi-bin/qgis_mapserv.fcgi
?SERVICE=WMS
&VERSION=1.3.0
&REQUEST=GetCapabilities
&map=/path/to/qgis/projects/nameofproject.qgs
  
```

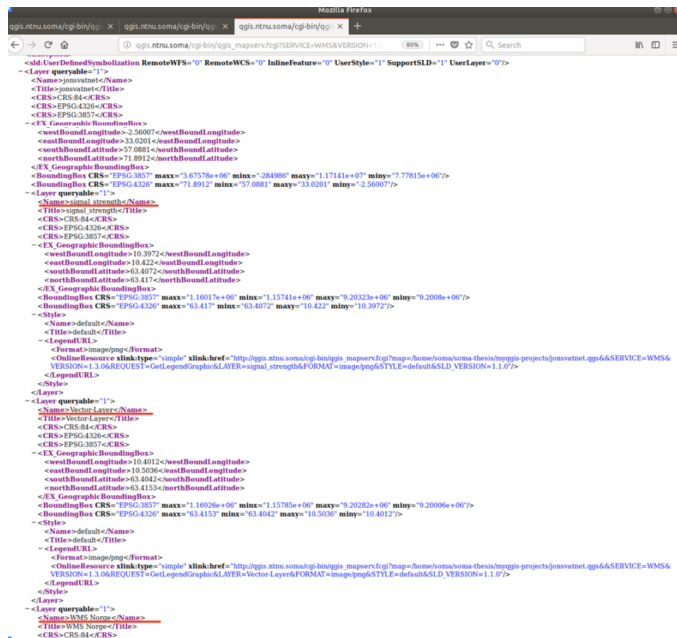


Figure 4.8: This figure is the image of the QGIS Server response in the form of XML Document to GetCapabilities HTTP Request

– WMS- GetMap Request

To have a response in the form of an image, instead of using GetCapabilities, GetMap Request is made by web-client to the QGIS Server as shown in Figure 4.9. The image display is static visualization of data and cannot be modified.

```

http://qgis.ntnu/cgi-bin/qgis_mapserv.fcgi
?MAP=/path/to/qgis/projects/nameofproject.qgs
&SERVICE=WMS
&VERSION=1.3.0
&REQUEST=GetMap
&BBOX=-432786,4372992,3358959,7513746
&SRS=EPSG:3857
&WIDTH=665
&HEIGHT=551
&LAYERS=jonsvatnet
&FORMAT=image/jpeg

```

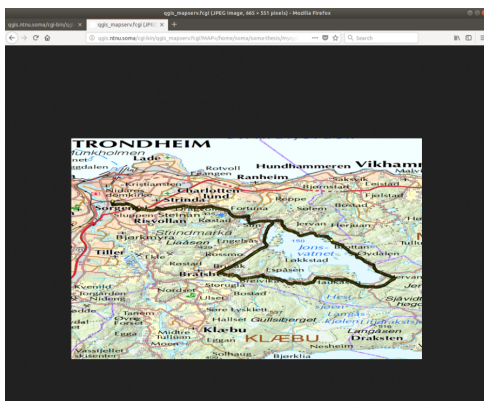


Figure 4.9: This figure is the Static image of the QGIS Server response to GetMap HTTP Request

In the above two requests the command line stating, "`?MAP=/path/to/qgis/projects/nameofproject.qgs`", this implies to the already created and stored project. This makes QGIS Server capable of producing only "Static" images meaning the data represented cannot be updated automatically. Due to this reason it is not suitable to use for this project where the major concern is creating flexible and dynamic visualization of spatiotemporal data. The only advantage of using QGIS Server over QGIS Desktop is that it can produce static visualization to view and analyze from anywhere on the globe and not restricted to the extent of the local platform. For achieving the dynamic visualization a standalone application is developed which is further described in detail in Chapter 7.

4.4 Brief Introduction to NTNU-QGIS Web API

A Web API stand for an Application Programming Interface for either a web server or a web browser. It is a web development concept, usually limited to a web application's client-side (including any web frameworks being used), and thus usually does not include web server or browser implementation details unless publicly accessible by a remote web application. - Wikipedia

The diagram Figure 4.10 below gives a brief illustration about the working of the central web API.

In addition to the above development of NTNU-QGIS web application incorporating three web pages, the web application also provides an Web API interface to all the external devices (Android/ios) which contain this application. This API gathers

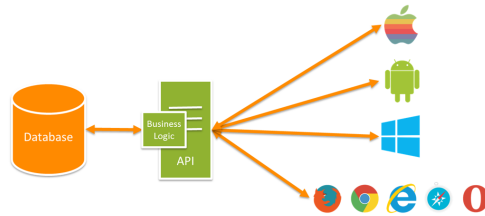


Figure 4.10: This image shows the central Web API Architecture holding all the business logic, adopted from [Mic]

all the data from its users and send it to the system. Instead of authorizing an user and doing log-in, simply an authorized device can directly upload data into the PostgreSQL database table for monitoring purposes. Data will be specific to particular device which is helpful to track details of an individual device.

4.5 Summary

The main purpose for developing this web application was to give an ease to the users to upload the spatiotemporal data in any file format, which can easily be pushed into created PostgreSQL database described in next Chapter 5. The updated file can be anytime re-uploaded which will automatically update the data stored inside the database. The web application is platform independent and consists of collection of responsive web pages therefore it is suitable for any device to open it and upload the data in whichever format.

The implementation of QGIS Server was an addition to provide an additional feature to the system but its static limitation makes it unsuitable to make further development using it with respect to this project. In the future work, the development of web API support is useful for providing direct way to connect with external devices.

Chapter 5

Spatial Database Setup

There is an innumerable definition for **data** but for our basic understanding we can say: A **data** is an information which can be in the form of images, audio/video clips, software programs, text documents, etc.

A **database** is basically a collection of various sort of data which is organized in a manner that make simple and easy access to the large data sets. Most databases contain data stored in the form of tables with different attributes information.

On the other hand, the **Database-Management System (DBMS)** is a software application which interacts with various other application, end-users and also with the database itself. A DBMS serves as an interface between the applications or end-users and the database facilitating the easy managing, storing, editing and analyzing of the data [Tech].

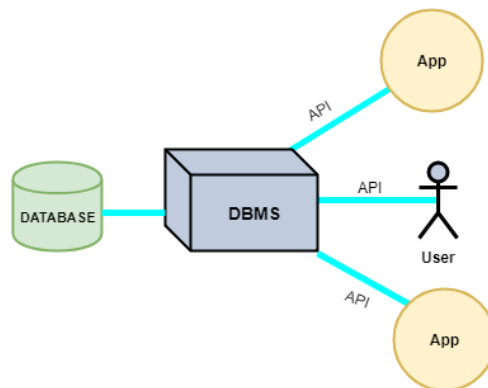


Figure 5.1: A Basic Architecture of Database System

Using a DBMS provides not only storage and easy data accessibility but also advantageous facilities like data backup, robust data integrity and data security which is

one of the topmost concern for everyone to protect their information. Some most widely used DBMS are PostgreSQL, MySQL, Microsoft Access, SQL Server, Oracle.

5.1 Why PostgreSQL?

PostgreSQL is a powerful and free open source Object-Relational Database Management System (ORDBMS) which is released under a BSD-style license¹. It has earned a strong reputation for providing reliability, data integrity, and its proven architecture to facilitate with performant and innovative solutions. The standard Structured Query Language (SQL) is the interface language used by PostgreSQL which allows storing, edit/update and to query the stored data inside the tables of the database. In addition to use, it also extends the capabilities of SQL language by combining it with many robust feature sets that not only safely store but also safely scale the most complicated data workloads [posb].

For this thesis, considering the formation of Spatial databases, the PostgreSQL is best suitable DBMS because it was one of the first databases to adopt the spatial databases. Being one of the first and extending its capabilities for easy handling of spatial objects by doing a major implementation of spatial functions by using **PostGIS** which is highly optimized for spatial queries. The further chapter describe in detail about PostGIS in Section 5.3.1 and about Spatial databases refer Section 2.1.2.

5.2 PostgreSQL Setup

Here, in this section we will go through all the process needed to implement our database including creation of tables and push the data into it which was gathered for visualization. As mentioned in Section 3.2.4, all the installation of the software is done under Ubuntu [Qgib].

5.2.1 PostgreSQL Installation

We need to first install the PostgreSQL by executing the following command on the terminal. The version of the PostgreSQL should be 9.x or latest. The development of the system has been done using Postgres version 9.6.

```
$ sudo apt-get install postgresql-9.6
```

5.2.2 Creating a Postgres Database User

After the successful installation of PostgreSQL, we need to become a Postgres user in order to use the DBMS.

¹it is commonly called as a permissive software license, which is a free software license with keeping nominal requirements on how the software be redistributed.

- This command below is used to become Postgres User.

```
$ sudo su - postgres
```

- After finishing the first step, it will take you to the postgres user's bash prompt. The below command is used to create a own database user account and later enter the password when prompted which is required for security purposes. For this thesis, **Database User Account : soma** is created, as the username should be matched with the unix-login name which will help in automatic authentication when a user is logged in.

```
$ createuser -d -E -i -l -P -r -s soma
```

The **-d -E -i -l -P -r -s** options are the various roles that can be assigned to the database user. By using this we can secure our database from other users who have access to the database by limiting there role by giving specific permission on using the data inside the database. The Figure 5.2 below gives details about various options a database user can be granted in a PostgreSQL database system.

```
-d, --createdb    role can create new databases
-E, --encrypted  encrypt stored password
-i, --inherit     role inherits privileges of roles it is a member of (default)
-l, --login      role can login (default)
-P, --pwprompt   assign a password to new role
-r, --createrole role can create new roles
-s, --superuser  role will be superuser
```

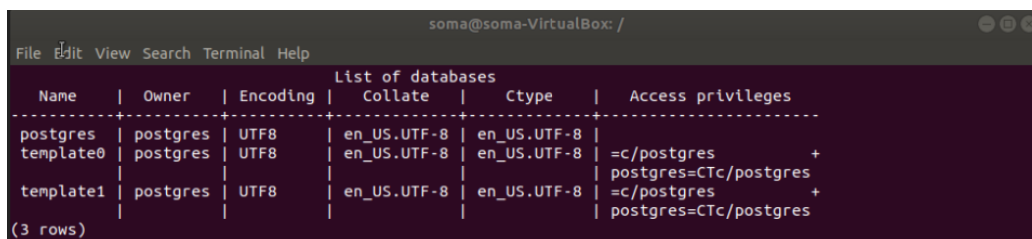
Figure 5.2: The Various PostgreSQL Permissions to Database User [Qgia]

- Verify the created new user account by running the following command below on the postgres user's bash prompt and the Figure 5.3 should be resulted.

```
$ psql -l
```

5.2.3 Creating a new Database

As mention before in the introduction of this chapter, the database is used for storing, creating/updating and analyzing the data. So we need a database for our system, which can store or create data for our project. This data in the PostgreSQL will be used further to analyze through visualization techniques. The visualization technique



List of databases					
Name	Owner	Encoding	Collate	Ctype	Access privileges
postgres	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	
template0	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres + postgres=CTc/postgres
template1	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres + postgres=CTc/postgres

(3 rows)

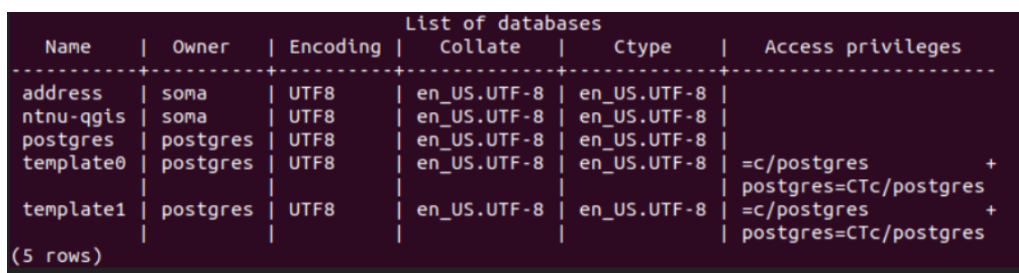
Figure 5.3: The image shows the screenshot image of Verification of becoming Postgres User

implies here as the various GIS technologies available but in this project we will be working with QGIS as described in Section 2.3. The process of creating new database are:

- The new database is created using **createdb** command. The below command should be run on the postgres bash prompt and later verify the existence of the created database under **soma** which is Postgres user account as shown in the Figure 5.4.

Created database name: **ntnu-qgis**

```
$ createdb ntnu-qgis -O soma
$ psql -l
```



List of databases					
Name	Owner	Encoding	Collate	Ctype	Access privileges
address	soma	UTF8	en_US.UTF-8	en_US.UTF-8	
ntnu-qgis	soma	UTF8	en_US.UTF-8	en_US.UTF-8	
postgres	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	
template0	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres + postgres=CTc/postgres
template1	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres + postgres=CTc/postgres

(5 rows)

Figure 5.4: Creation of **ntnu-qgis** Database

5.2.4 Creating a Data Table

As we are dealing with Norwegian road traffic data which have information about car objects containing geographic coordinates as well as cellular network information.

So we need to create a table which will store all these attributes information as presented in user input file inside a database.

Under Section 2.4, the Android application generates records containing various car attributes information. After a particular time interval, a new position is captured by the application and added as a new record in the data. When the application finishes capturing information related to a particular tour, the entire data captured by the device is converted to a Comma Separated Values (CSV) file format. Henceforth, it is pushed into the database table using the NTNU-QGIS webpage (refer Section 4.2.1.3).

There are two ways of making a table either using SQL in postgres bash prompt or by using the created postgres GUI as later described in Section 5.2.4.2.

5.2.4.1 Making a table using Postgres-Bash Prompt

- Firstly, connect to the created database **ntnu-qgis** by typing the following command:

```
$ psql ntnu-qgis
```

- Creating a table using **create table** command. For this project, table of name **route** is created and the table schema² are the attributes according to the header field in user input file. The view of the table schema is shown in Figure 5.5 below.

```
ntnu-qgis=# \d route
```

Column	Type	Modifiers
id	integer	not null default nextval('route_id_seq'::regclass)
time	numeric	
communication_protocol	character varying(50)	
round_trip_delay	numeric	
signal_strength	numeric	
latitude	numeric	
longitude	numeric	
altitude	numeric	
bearing	numeric	
speed	numeric	
accuracy	numeric	
total_satellites	integer	
satellites_in_fix	integer	

```
Indexes:
    "route_pkey" PRIMARY KEY, btree (id)
```

Figure 5.5: The view of **route** table schema

²is the organization of data as how it is logically structured inside the database meaning a column which is linked to one particular data-type

5.2.4.2 Alternate option to create table : Postgres GUI

Everyone is not comfortable in creating a table using SQL, so the best alternate way to create and manage data inside the table is using postgres user interface. After setting up the Apache web server as mention in Section 4.1, we can open Postgre-GUI from the browser to create, alter or manage the tables much easily. Run the following command to install it:

```
$ psql apt-get install phpPgAdmin
```

The thing which is important here to notice is that, depending on operating system 's version the command is also modified by mentioning the php version while using the command. The Figure 5.6 below gives the view of the postgresQL GUI called pgAdmin which is however a simpler and quicker way to create, edit, drop tables by click of a mouse. The previous created table implemented from bash prompt using SQL Section 5.2.4.1 can be seen on the GUI.

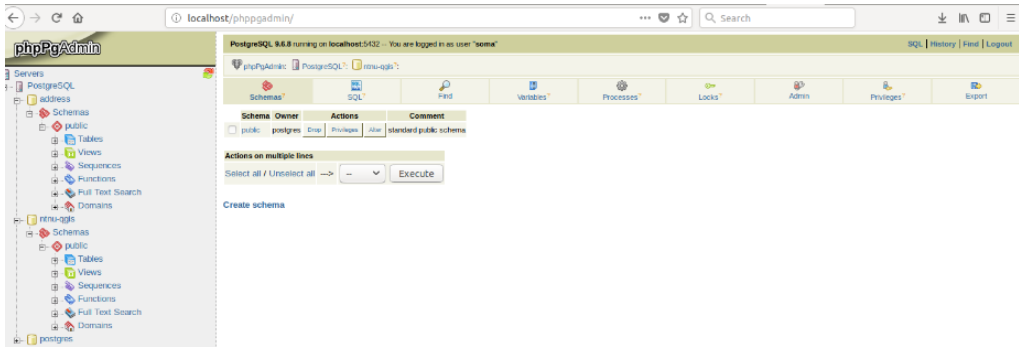


Figure 5.6: The view of pgAdmin GUI on Apache- Localhost

5.3 PostGIS: Creating Spatiotemporal Databases

Conventionally, databases were designed to organize and structure data of any format. However, as the size of the database is increasing and to optimize database in the Geospatial domain we use **Spatial Databases**. In order words, the Spatial database arises when it has to manages and deal with data which is collected across space as well as time as previously described under Section 2.1.2.

Using Spatial Databases allows the storing of the geometries of records refer to as **Geospatial objects** inside a Database. In addition, it provides functionality to retrieve and query the records with improved response time on these stored heavy geometries by creating light-weight spatial index (similar to hashMaps in Java

domain). These Geometries can be of different shapes in the form of points, line or polygons and the requirements evolve as the size of the data increases depending on the scenario where Spatial Database can limit the potential use by making a constraint.

The one most prominent example of spatial databases are the Satellite images. As we know spatial objects are complex (Section 2.1.3), so to extract the spatial information out from the satellite images, it needs to be processed on the spatial frame of reference where 3D images are converted to 2D images will be described in more detail later under Section 6.2. However, another most prominent example of spatial databases are Maps. Maps objects are also stored in the Spatial databases [Spr].

This forthcoming sections provides a detail description on:

- **Section 5.3.1 : What is PostGIS?**
- **Section 5.3.2 : Setting Up PostGIS**
- **Section 5.3.3 : Testing PostGIS Functionality**

5.3.1 What is PostGIS?

PostGIS extends PostgreSQL with robust spatial database management capabilities which allows Geographic Information System (GIS) objects to be stored in the database [posa] [Bou].

PostGIS is freely available, and fairly an Open Geospatial Consortium (OGC) compliant software and being used as an extender for PostgreSQL, which is a form of Object-Relational Database Management System (ORDBMS). While PostGIS is a *free and open source software*, it is used on both GIS sectors, public (e.g., QGIS) and commercial (e.g., ArcGIS). In PostgreSQL with this PostGIS extension brings the most comprehensive geo-functionalities with over one thousand in-built spatial functions (Section 6.2) which enhances the handling of spatial data within relational database structure [Alt].

Advantageous Features for PostGIS:

- The language of PostGIS is similar to SQL which is the also an common interface language used by PostgreSQL which allows performing complex queries and spatial analysis on the spatial data with an ease.
- It supports different types of geometries like Lines, Points, Polygons, LineStrings, etc.

- PostGIS supports spatial indexes³ for making fast spatial queries on the big databases.
- As the implementation of PostGIS is based on **light-weight** indexes and geometries which optimizes the use of disk memory for storing heavy data.
- The analytical and various processing function are simply performed within PostGIS on the vector and raster data allowing an easy generation of maps view which have the desired analytical output.

5.3.2 Setting Up PostGIS

Implementation of PostGIS functionality will provide the access to the spatial functions from within PostgreSQL.

5.3.2.1 Installation of PostGIS

Run the following command on the terminal using the already installed version of PostgreSQL.

```
$ sudo apt-get install postgis
$ sudo apt-get install postgresql-9.6-postgis
```

5.3.2.2 Implementation of PostGIS on ntnu-qgis Database

When the PostGIS is installed we need to configure our database to use the extensions provided by the PostGIS. The figure below shows the configuration of PostGIS on our database **ntnu-qgis** which makes it geo-spatially enabled. In the upcoming sections it will be more clear about what does PostGIS offer and why its a requirement in spatial domain.

```
sona@sona-VirtualBox:/$ psql -d ntnu-qgis -c "CREATE EXTENSION postgis;"
CREATE EXTENSION
sona@sona-VirtualBox:/$ █
```

Figure 5.7: Configuration of PostGIS on **ntnu-qgis** Database

5.3.3 Testing PostGIS Functionality

PostGIS basically is a collection of in-built database functions which extends the core capabilities of PostgreSQL so that it can store, retrieve, query and alter spatial

³is a type of extended light-weight index which allows to index a spatial column, to improve performance on spatial query over spatial feature

data. In order to make this happen, various different functions are installed into the database [qgis].

Now as our database "ntnu-qgis" is geo-spatially enabled which means that our database has the functionality to create the various forms of spatial objects on the map according to the various records present in the data gathered.

In this thesis, we will be only considering **Point Function** as we are dealing with coordinate geometries acquired from road traffic data where each records in data implies to Point car object. Run this below command from the Postgres bash prompt after connecting to "ntnu-qgis" database which will show all the functions related to Point geometries as shown in Figure 5.8:

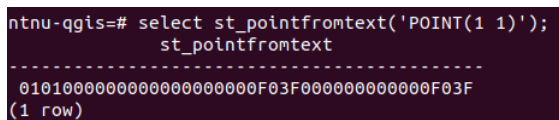
```
\df *point*
```

Schema	Name	Result data type	Argument data types
pg_catalog	dist_cpoint	double precision	circle, point
pg_catalog	gist_point_compress	internal	internal
pg_catalog	gist_point_consistent	boolean	internal, point, smallint, oid
pg_catalog	gist_point_distance	double precision	internal, point, smallint, oid
pg_catalog	gist_point_fetch	internal	internal

Figure 5.8: PostGIS Point function on ntnu-qgis Database

There will be list of other available functions related to Point geometry but we test using command `st_pointfromtext` [qgis]. After executing the command, it will show the result as depicted in Figure 5.9 of the testing of PostGIS functionality based on text based "Point" function. The resulted row is in the OGC format called as Well-Known Binary (WKB) described in next chapter under Section 6.2:

```
select st_pointfromtext('POINT(1 1)');
```



```
ntnu-qgis=# select st_pointfromtext('POINT(1 1)');
          st_pointfromtext
-----
010100000000000000000000F03F000000000000F03F
(1 row)
```

Figure 5.9: PostGIS Point function testing at Position 1,1

Here, the main purpose of testing Point function is to demonstrate the complex structure of these spatial objects. This text (record) contained in the data is converted into spatial form to make it in a QGIS understandable format. Later, this is used in visualization process to present these complex form as a Point shape objects displayed on Maps in order to make a humanly understandable presentation.

- The command states that, a visual point on the map is created at position 1,1 using POINT (1,1) and assuming standard coordinate projection **EPSG:4326**⁴. (The **EPSG** stands for **European Petroleum Survey Group** and in detail described in next Section 6.2.)
- Here position 1,1 implied to geometric fields, x and y values which in spatial terms means longitude and latitude respectively to located accurately over maps. (To understand how it is performed refer Section 6.1)
- The Figure 5.10 below is a reference example to bring some clarity on how this point function is used in real life to see positioning of objects on the map-view.

⁴<http://www.epsg.org>



Figure 5.10: An example reference image showing how **PostGIS- Point function** is used to display location using data records. *source: mrbool.com/how-to-track-users-registration-location-in-google-maps/33562*

The next Section 6 gives the detail description on, how the point objects accurately placed over various geographical maps just with position coordinates and visualized over flat surface like maps.

Chapter 6

Spatial Reference System in Visualization

In addition to the various PostGIS functions, the extension also provide with a collection of Spatial Reference System (SRS) definitions which are defined by the European Petroleum Survey Group (EPSG) which helps in the Coordinate Reference System (CRS) conversions. These SRS definitions are stored in our database after installation of PostGIS. But what is SRS? and how this work? In the previous Chapter 5 we have seen how the spatial function "Point" is used to create a point object using the position 1,1 and projection EPSG: 84. This chapter gives a better understanding on how the coordinates system is used by PostGIS over PostgreSQL and, also how various forms of geometries are formed using values from our spatial dataset which are accuracy visualized on the map surface.

This chapter comprises of three sections:

- **Section 6.1 : What is Spatial Reference System (SRS) or Coordinate Reference Systems (CRS)?**
- **Section 6.2 : Implementation of Storing and Representation of Geographical Features in Database**
- **Section 6.3 : Summary**

6.1 What is Spatial Reference System (SRS) or Coordinate Reference Systems (CRS)?

A spatial reference system (SRS) or coordinate reference system (CRS) is a coordinate-based local, regional or global system used to locate geographical entities. – Wikipedia

When any object has to be located we often refer to a "Coordinate System". This system uses X and Y values (and sometimes Z values) in order to specify location of the objects within a 2D or 3D space domain as shown in Figure 6.1.

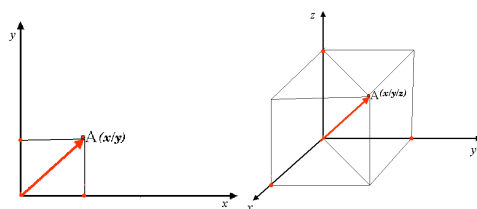


Figure 6.1: This image shows a basic Coordinate System known as Cartesian system. Source: <https://svn.osgeo.org/qgis/docs/>

We live in a 3-Dimensional plane, Earth which is considered as *oblate spheroid*, or *oblate ellipsoid* in shape. To locate any object accurately on the Earth which is an approximately a "round" figure, we need to follow a coordinate system that adapts to the shape of the Earth. When we define map on a paper or any flat surface, we move from 3-Dimensional space (Globe) to a 2-Dimensional space, this conversion in geo-spatial domain is define by CRS. The components of CRS is used to define the translation between the data that exists in a 3-Dimensional space (a round-Earth) to on a "Flattened" 2-Dimensional Coordinate system [oCa]. The reference Figure 6.2 gives an overview of how the center of the Earth, called Origin is transformed and placed from the round-3D plane onto a flat map surface.

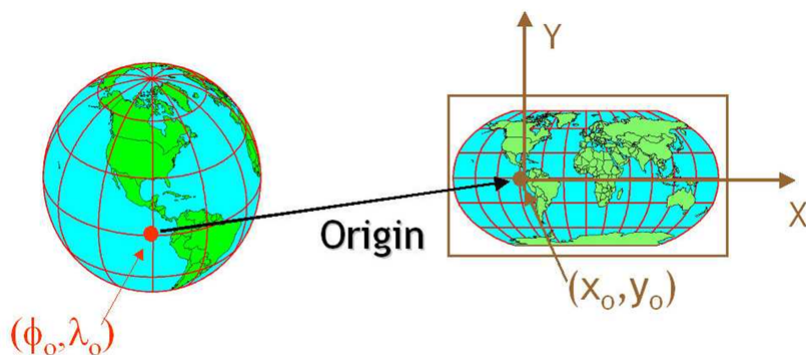


Figure 6.2: A CRS defined conversion of location on a 3-Dimensional plane to a 2-Dimensional plane. Source: <http://ayresriverblog.com>

Some key components of CRS:

- **Coordinate System:** The X,Y or Z grid to define the location of data in space.
- **Horizontal and vertical units:** These units are used to define the grid along the x,y or z axes.

- **Projection Information:** This is a mathematical equation used for flattening the objects which exists on a 3-Dimensional plane (e.g. Earth), so that they can be viewed on a flat surface (paper maps, QGIS map canvas, etc.)
- **Datum:** This is a modeled version of the Earth shape that defines the origin for the placing of coordinate system in space.

There are two types of Coordinate Reference System (CRS) as shown in Figure 6.3:

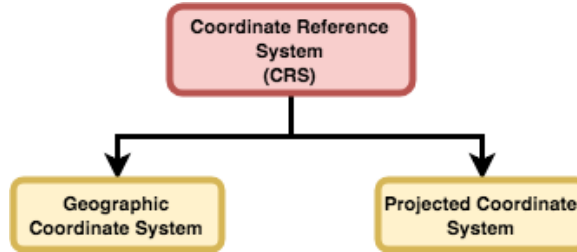


Figure 6.3: The two types of CRS

- **1) Geographic Coordinate Systems:**
This coordinate systems basically stretches to the extend of the entire globe (e.g. Latitude, Longitude). Below Figure 6.4 is a reference example showing the three coordinate locations on a 2-Dimensional Map using one of the CRS component "**Coordinate System**". (**Note:** the UNITS are in Decimal Degrees (Latitude/Longitude):

Oslo, Norway: 59.95001, 10.75003
Boulder, Colorado: 40.02744, -105.25195
Mallorca, Spain: 39.61674, 2.983333
- **2) Projected Coordinate Systems:** This coordinate system is localized to a particular region (e.g. UTM, State Plane) on the globe to **minimize the visual distortion**. This is further described in next Section 6.1.1

6.1.1 Projected Coordinate Systems

From above section, one of the type of CRS, **Geographic Coordinate Systems** are mainly used for creating global maps as it span over the entire globe using Latitude/Longitude. However, when it comes to quantifying scale or distance they are prone to errors. To overcome this limitation, various spatial projections have been developed gradually that can be used to more precisely define and capture shape, scale/distance and area of any spatial objects on the map [oCc].

This coordinate system uses spatial projection to minimize the distortion to provide more accuracy in locating objects by optimizing the relative shape and size of a

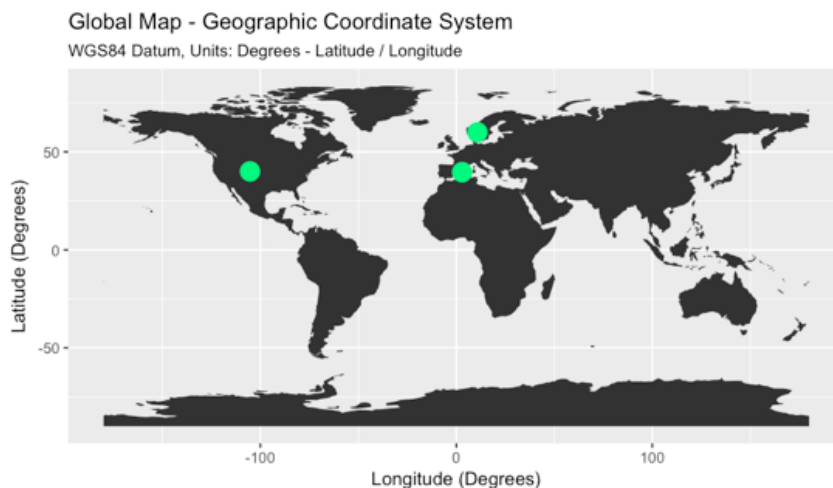


Figure 6.4: A CRS defined three Coordinate locations (Oslo, Boulder and Mallorca) on a 2D Map. *Source: <https://www.earthdatascience.org/>*

particular region on the globe. By the word "region" it implies to using either Universal Transverse Mercator (UTM), Robinson or, State plane. Spatial projection refers to as the mathematical calculations which are performed to translate the data existed in the 3-D plane onto a 2-D plane by flatten the data. During this conversion process to provide result in visualization format on the surface map, some area is expanded and some are compressed. The Figure 6.5 show the translation of 3-dimensional globe onto a flat surface 2-Dimensional map, demonstrating the various projections performed eventually bring changes in the appearance of the round figure.

Note: The **Universal Transverse Mercator (UTM)** system is normally used for spatial projection coordinate system. UTM divided the entire globe into region (North and South), and Zones which are numbered 0-60 which is equivalent to Longitude. The default origin (0,0) and region for each UTM zone is located at the intersection of the Equator.

To rephrase **Datum** again in the coordinate systems, it describes the vertical and the horizontal reference point to optimize the scale on the Map.

- The **Vertical Datum**, states the relationship between the Earth centre and a specific "ellipsoid" which is refer to as the top of the earth's surface.
- The **Horizontal Datum** defines the Longitude.

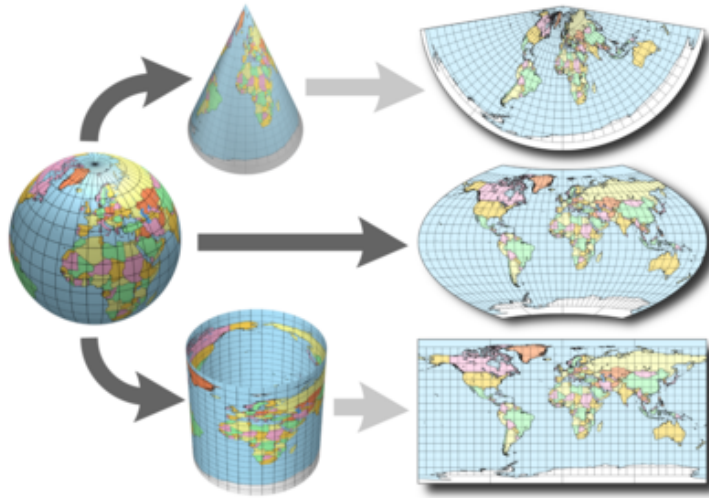


Figure 6.5: A reference figure with CRS translation of 3-Dimensional Globe to a 2-Dimensional flat surface Map. *Source: CA Furuti, progonos.com/furuti*

- The datum also defines the **origin (0,0)** the "center of Earth" of a coordinate system.

6.1.2 Coordinate Reference System (CRS)- Formats

To store a CRS information there exists numerous formats. The commonly used are Well-Known Text (WKT), Proj4 and EPSG. Our GIS tool- QGIS gathers all the CRS information after certain translations which is in the form of string. The string contains all of the CRS components which QGIS needs for accurate projections. Each components is specifies using a "+" sign [oCb].

In this project, we will be using European Petroleum Survey Group (EPSG) format for CRS. The EPSG codes are 4-5 digit number code that represents a particular CRS information. List of ESPG code are available on <http://spatialreference.org/ref/epsg/>.

An example of an EPSG string:

```
proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs
```

When the string is broken down into individual components of CRS, it is defined as follows:

- **+proj=longlat:** the data are in a geographic coordinate system (Latitude and Longitude)

- **+ellps=WGS84**: the ellipsoid (refers to as how the earth's roundness is determined), which is WGS84 for this data.
- **datum=WGS84**: the default value of datum is taken, WGS 84 refer as origin (0,0)

6.2 Implementation of Storing and Representation of Geographical Features in Database

As from previous sections we have learned how the spatial dataset are located accurately on the flat Map surface by forming a string using CRS components. Here, we see its implementation on our database. In order words, the database created contains data stored in the form records in a table which only saves the data loaded by the users file. This table has no spatial field configured yet which can be used to draw spatial objects to view on the map.

This section gives will describe the various forms of geometries exists and how it is configured into our "ntnu-qgis" database, so that we can plot the spatial objects using the information gathered and visualize precisely in order to fetch out meaningful information when data is queried. Before going into the detail we should know about OGC which stands for **Open Geospatial Consortium**.

The OGC's Simple feature model defines a Spatial Reference System using WKT, and the support has been implemented using several standards-based Geographic Information System (GIS) e.g. QGIS, ArcGIS.

The Open Geospatial Consortium (OGC), an international voluntary consensus standards organization, originated in 1994. In the OGC, more than 370+ commercial, governmental, nonprofit and research organizations worldwide collaborate in an open consensus process encouraging development and implementation of standards for geospatial content and services, GIS data processing and data sharing. - Wikipedia

6.2.1 Simple Feature Model for SQL

The Simple Feature for SQL (SFS) Model is a *non-topological* way for storing and defining function for accessing, creating and operating the geospatial data in a database. The Figure 6.6 show the representation model defining various geospatial objects like Point, Lines, Polygons etc. from spatial reference system [qgid].

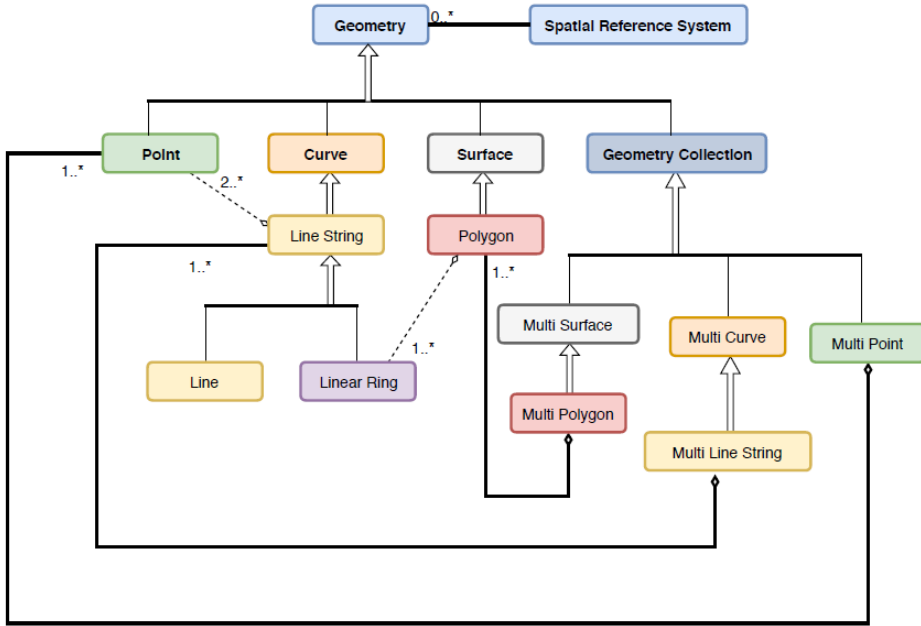


Figure 6.6: A Simple Feature for SQL (SFS) Model defining various geo-spatial data from Point, Lines, and Polygons

Firstly, let's see how the schema appears for a spatial reference system in a database. Type the following command and it will result something like as shown in Figure 6.7:

```
\d spatial_ref_sys
```

As per the schema in the figure, we see there are four columns holding various details about various projections. The *srid* column contains the various European Petroleum Survey Group (EPSG) codes while the *proj4text* column is the complete CRS strings including its various components. By default, spatial reference check is implemented where *srid* cannot be NULL otherwise without it spatial objects will not be given any projection to be visible over the map.

```

ntnu-qgis=# \d spatial_ref_sys
Table "public.spatial_ref_sys"
  Column          |          Type          | Modifiers
-----+-----+-----
srid              | integer               | not null
auth_name        | character varying(256)
auth_srid        | integer
srtext           | character varying(2048)
proj4text        | character varying(2048)
Indexes:
"spatial_ref_sys_pkey" PRIMARY KEY, btree (srid)
Check constraints:
"spatial_ref_sys_srid_check" CHECK (srid > 0 AND srid <= 998999)

```

Figure 6.7: A schema of Spatial Reference System Table

We can also see the entire table in Table 6.1 with all the possible reference system available with the SRID string used for projection during spatial visualization.

Table 6.1: A table contain various SRID

srld	auth_name	auth_srid	srtext	proj4text	
226	4297	EPSG	4297	GEOGCS["Tanarive",DATUM["Tanarive_1925",SPHEROID["International 1924",6378388,297,AUTHORITY["EPSG","7022"]],TOWGS84[-189,-242,91,0,0,0],A...	+proj=longlat +ellps=intl +tows84=-189,-242,...
227	4298	EPSG	4298	GEOGCS["Timbalal 1948",DATUM["Timbalal_1948",SPHEROID["Everest 1830 (1967 Definition)",6377298.556,300.8017,AUTHORITY["EPSG","7016"]],TOWGS84[...	+proj=longlat +ellps=evrst55 +tows84=-679,6...
228	4299	EPSG	4299	GEOGCS["TM65",DATUM["TM65",SPHEROID["Airy Modified 1849",6377340.189,299.3249646,AUTHORITY["EPSG","7002"]],TOWGS84[482.5,-130.6,564.6,-1.042,...	+proj=longlat +datum=ire65 +no_defs
229	4300	EPSG	4300	GEOGCS["TM75",DATUM["Geodetic Datum of 1965",SPHEROID["Airy Modified 1849",6377340.189,299.3249646,AUTHORITY["EPSG","7002"]],TOWGS84[482.5,-130.6,564.6,-1.042,...	+proj=longlat +ellps=bessel +no_defs
230	4301	EPSG	4301	GEOGCS["Tokyo",DATUM["Tokyo",SPHEROID["Bessel 1841",6377397.155,299.1528128,AUTHORITY["EPSG","7004"]],TOWGS84[146.414,597.337,680.507,0.0,0.0,...	+proj=longlat +ellps=bessel +tows84=-146.41...
231	4302	EPSG	4302	GEOGCS["Trinidad 1903",DATUM["Trinidad_1903",SPHEROID["Clarke 1858",6378293.645208759,294.2606763692606,AUTHORITY["EPSG","7007"]],TOWGS84[...	+proj=longlat +a=6378293.645208759 +b=635...
232	4303	EPSG	4303	GEOGCS["TC(1948)",DATUM["Trucial Coast_1948",SPHEROID["Helmer 1906",6378200,298.3,AUTHORITY["EPSG","7020"]],AUTHORITY["EPSG","6303"],PRIME...	+proj=longlat +ellps=helmer +no_defs
233	4304	EPSG	4304	GEOGCS["Voiron 1875",DATUM["Voiron_1875",SPHEROID["Clarke 1880 (IGN)",6378249.2,293.4660212936269,AUTHORITY["EPSG","7011"]],TOWGS84[73,-247,2...	+proj=longlat +a=6378249.2 +b=6356515 +tow...
234	4306	EPSG	4306	GEOGCS["Bern 1938",DATUM["Bern_1938",SPHEROID["Bessel 1841",6377397.155,299.1528128,AUTHORITY["EPSG","7004"]],AUTHORITY["EPSG","6306"],PR...	+proj=longlat +ellps=bessel +no_defs
235	4307	EPSG	4307	GEOGCS["Nord Sahara 1959",DATUM["Nord_Sahara_1959",SPHEROID["Clarke 1880 (RGS)",6378249.145,293.465,AUTHORITY["EPSG","7012"]],TOWGS84[209.3,...	+proj=longlat +ellps=clrk80 +tows84=-209.36...
236	4308	EPSG	4308	GEOGCS["RT38",DATUM["Stockholm_1938",SPHEROID["Bessel 1841",6377397.155,299.1528128,AUTHORITY["EPSG","7004"]],AUTHORITY["EPSG","6308"],PR...	+proj=longlat +ellps=bessel +no_defs
237	4309	EPSG	4309	GEOGCS["vacare",DATUM["Vacare",SPHEROID["International 1924",6378388,297,AUTHORITY["EPSG","7022"]],TOWGS84[-155,171,37,0,0,0],AUTHORITY["EPS...	+proj=longlat +ellps=intl +tows84=-155,171,3...
238	4310	EPSG	4310	GEOGCS["Yoff",DATUM["Yoff",SPHEROID["Clarke 1880 (IGN)",6378249.2,293.4660212936269,AUTHORITY["EPSG","7011"]],TOWGS84[30.0,189,0,0,0,0],AUTH...	+proj=longlat +a=6378249.2 +b=6356515 +tow...
239	4311	EPSG	4311	GEOGCS["Zanderij",DATUM["Zanderij",SPHEROID["International 1924",6378388,297,AUTHORITY["EPSG","7022"]],TOWGS84[265.120,-358.0,0,0,0],AUTHORITY[...	+proj=longlat +ellps=intl +tows84=-265,120,...
240	4312	EPSG	4312	GEOGCS["MG",DATUM["Militar_Geographische_Institute",SPHEROID["Bessel 1841",6377397.155,299.1528128,AUTHORITY["EPSG","7004"]],TOWGS84[577.32...	+proj=longlat +datum=hermannskogel +no_defs
241	4313	EPSG	4313	GEOGCS["Belge 1972",DATUM["Reseau_National_Belge_1972",SPHEROID["International 1924",6378388,297,AUTHORITY["EPSG","7022"]],TOWGS84[-106.8686,...	+proj=longlat +ellps=intl +tows84=-106.8686,...
242	4314	EPSG	4314	GEOGCS["DHDN",DATUM["Deutsches_Hauptdreiecksnetz",SPHEROID["Bessel 1841",6377397.155,299.1528128,AUTHORITY["EPSG","7004"]],TOWGS84[598.17,1...	+proj=longlat +datum=potsdam +no_defs
243	4315	EPSG	4315	GEOGCS["Conakry 1905",DATUM["Conakry_1905",SPHEROID["Clarke 1880 (IGN)",6378249.2,293.4660212936269,AUTHORITY["EPSG","7011"]],TOWGS84[-23.2,...	+proj=longlat +a=6378249.2 +b=6356515 +tow...
244	4605	EPSG	4605	GEOGCS["St. Kitts 1955",DATUM["St_Kitts_1955",SPHEROID["Clarke 1880 (RGS)",6378249.145,293.465,AUTHORITY["EPSG","7012"]],TOWGS84[9.183,236.0,0,0,0,...	+proj=longlat +ellps=clrk80 +tows84=9.183,2...
245	4316	EPSG	4316	GEOGCS["Dealul Piscului 1930",DATUM["Dealul_Piscului_1930",SPHEROID["International 1924",6378388,297,AUTHORITY["EPSG","7022"]],TOWGS84[103.25,10...	+proj=longlat +ellps=intl +tows84=103.25,10...
246	4317	EPSG	4317	GEOGCS["Dealul Piscului 1970",DATUM["Dealul_Piscului_1970",SPHEROID["Krasowsky 1940",6378245,298.3,AUTHORITY["EPSG","7024"]],TOWGS84[28,-121,...	+proj=longlat +ellps=krass +tows84=28,-121,...
247	4318	EPSG	4318	GEOGCS["NGN",DATUM["National_Geodetic_Network",SPHEROID["WGS 84",6378137,298.257223563,AUTHORITY["EPSG","7030"]],TOWGS84[-3.2,-5.2,8.0,0.0,0,0,...	+proj=longlat +ellps=WGS84 +tows84=-3.2,-5.2...
248	4319	EPSG	4319	GEOGCS["KUDAMS",DATUM["Kuwait_Ubility",SPHEROID["GRS 1980",6378137,298.257222101,AUTHORITY["EPSG","7019"]],TOWGS84[-20.8,11.3,2.4,0.0,0,0],AU...	+proj=longlat +ellps=GRS80 +tows84=-20.8,1...
249	4322	EPSG	4322	GEOGCS["WGS 72",DATUM["WGS_1972",SPHEROID["WGS 72",6378135,298.26,AUTHORITY["EPSG","7043"]],TOWGS84[0,0,4.5,0,0,0,554,0,2263],AUTHORITY["E...	+proj=longlat +ellps=WGS72 +tows84=0,0,4.5,...
250	4324	EPSG	4324	GEOGCS["WGS 72BE",DATUM["WGS_1972_Transit_Broadcast_Ephemeris",SPHEROID["WGS 72",6378135,298.26,AUTHORITY["EPSG","7043"]],TOWGS84[0,0,1.9,...	+proj=longlat +ellps=WGS72 +tows84=0,0,1.9,...
251	4326	EPSG	4326	GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"],PRIME[M...]	+proj=longlat +datum=WGS84 +no_defs
252	4463	EPSG	4463	GEOGCS["RCSPM06",DATUM["Reseau_Geodesique_de_Saint_Pierre_et_Miquelon_2006",SPHEROID["GRS 1980",6378137,298.257222101,AUTHORITY["EPSG","...]]],TOWGS84[...	+proj=longlat +ellps=GRS80 +tows84=0,0,0,0,...
253	4470	EPSG	4470	GEOGCS["RCM04",DATUM["Reseau_Geodesique_de_Mayotte_2004",SPHEROID["GRS 1980",6378137,298.257222101,AUTHORITY["EPSG","7019"]],TOWGS84[0,0,0,0,0,0,...	+proj=longlat +ellps=GRS80 +tows84=0,0,0,0,...
254	4475	EPSG	4475	GEOGCS["Cadastre 1997",DATUM["Cadastre_1997",SPHEROID["International 1924",6378388,297,AUTHORITY["EPSG","7022"]],TOWGS84[-381.788,-57.501,256,...	+proj=longlat +ellps=intl +tows84=-381.788,...
255	4483	EPSG	4483	GEOGCS["Mexico ITRF92",DATUM["Mexico_ITRF92",SPHEROID["GRS 1980",6378137,298.257222101,AUTHORITY["EPSG","7019"]],TOWGS84[0,0,0,0,0,0,0],AUT...	+proj=longlat +ellps=GRS80 +tows84=0,0,0,0,...

In this thesis, as we are designing a system to visualize spatiotemporal data in real-time so the best approach to locate the object in the dataset is by using **Coordinate Reference System**. As each CRS is optimized to give the best representation of the shape, scale and area of features in a data-set. The coordinate system includes **Latitude** and **Longitude**, units are **Decimal Degrees** and the Datum used is **WGS84**¹.

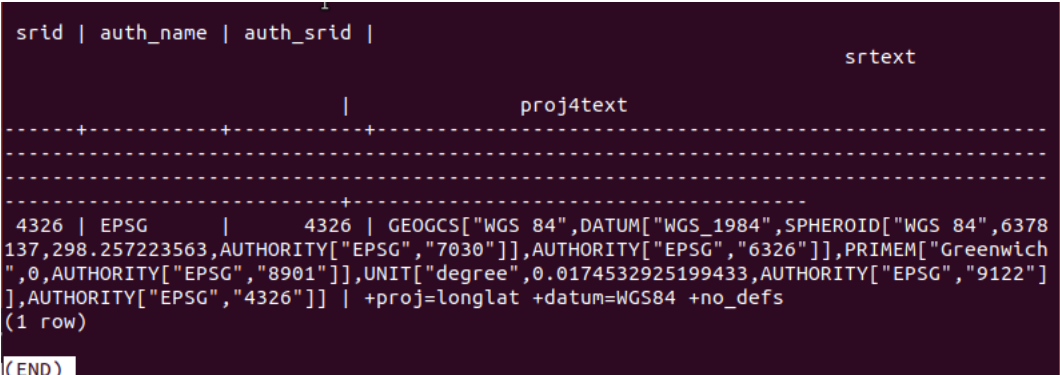
¹is a standard U.S. Department of Defense definition created in 1984 for the use in geodesy, cartography and for navigation(including GPS)

The World Geodetic System (WGS) 84 is frequently used Datum and referred to as "The origin" which is the center of the Earth..

To see how this work, we can do a quick check on one of the SRID which we are interested in using for this work is EPSG:4326 - the geographic lat/lon reference system using the WGS 84 ellipsoid. Run the following SQL query command below and it will result something like in Figure 6.8:

```
select * from spatial_ref_sys where srid=4326;
```

A Spatial Reference System Identifier (SRID) is a unique value used to unambiguously identify projected, unprojected, and local spatial coordinate system definitions. These coordinate systems form the heart of all GIS applications. – Wikipedia



```

srid | auth_name | auth_srid |
-----+-----+-----+-----
              |          |          | proj4text
-----+-----+-----+-----
4326 | EPSG      |    4326 | GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378
137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich
",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]
],AUTHORITY["EPSG","4326"]] | +proj=longlat +datum=WGS84 +no_defs
(1 row)
(END)
```

Figure 6.8: EPSG:4326 - the geographic lat/lon reference system using the WGS 84 ellipsoid

6.2.2 Adding spatial geometry field to table

The created "route" table as mention in Section 5.2.4.1, is a simple PostgreSQL table but as we have installed additional functionality i.e. PostGIS which facilitates to make spatial geometries to be stored into a database. We have discussed two ways of creating and modifying the data fields inside a database either by using PostgreSQL bash shell prompt or by pgAdmin GUI Section 5.2.4.

The Figure 6.9 shows the addition of geometry field using SQL bash prompt which later will be seen on pgAdmin GUI.

```
ntnu-qgis=# alter table route add column the_geom geometry;
ALTER TABLE
ntnu-qgis=#
```

Figure 6.9: A SQL command to Alter "route" table by adding Geometry field

As for our project, we are only interested in one type of geometry "Point", so we add a constraint on our table that it will only accept Point geometries from data-set. The Figure 6.10 below show how to perform this Point constraint and the only advantage of doing this is to narrow the dataset entry to the table for better systematic storage of same type of data during visualization which will in way increase performance. The Figure 6.11 below shows how the addition of geometry field in the table "route" with constraint appear.

```
ntnu-qgis=# alter table route add constraint route_geom_point_chk check(st_geometrytype
(the_geom) = 'ST_Point'::text OR the_geom IS NULL);
ALTER TABLE
ntnu-qgis=#
```

Figure 6.10: Performing Point Constraint on table "route"

```
ntnu-qgis=# \d route
```

Column	Type	Modifiers
id	integer	not null default nextval('route_id_seq'::regclass)
communication_protocol	character varying	
round_trip_delay	integer	
signal_strength	integer	
latitude	numeric	
longitude	numeric	
altitude	numeric	
bearing	numeric	
speed	numeric	
accuracy	numeric	
total_satellites	numeric	
satellites in fix	numeric	
the_geom	geometry	

```
Indexes:
  "route_pkey" PRIMARY KEY, btree (id)
  "sidx_route_the_geom" gist (the_geom)
Check constraints:
  "route_geom_point_chk" CHECK (st_geometrytype(the_geom) = 'ST_Point'::text OR the_geom IS NULL)
```

```
ntnu-qgis=#
```

Figure 6.11: An addition of Geometry field with "Point" constraint on table "route"

After performing the above steps, now when the data of any format is uploaded using uploading web page, the database will quickly add the coordinate system columns and convert in into spatial format and place records in the "the_geom" column created for this purpose. More specifically, because of the implementation of PostGIS functionality, the coordinate fields (Latitude and Longitude) are directly converted

6.2. IMPLEMENTATION OF STORING AND REPRESENTATION OF GEOGRAPHICAL FEATURES IN DATABASE 57

to a OGC format using SRID:ESPG 4326. There was a limitation in QGIS desktop where everything has to be done manually but now the user when using this database will set everything automatically to create spatial objects which can be visualized on the Map.

To bring more clarity, The Figure 6.12 below shows the cellular road traffic data uploaded by user with latitude and longitude values in the table "route".

```
ntnu-qgis=# select longitude, latitude from route ;
 longitude | latitude
-----+-----
 10.39961234 | 63.41656451
 10.39957537 | 63.41655563
 10.39952299 | 63.41654083
(3 rows)
```

Figure 6.12: A Geometry fields with Latitude and Longitude object values

The next Figure 6.13 shows the conversion of latitude and longitude coordinate fields automatically by the database now in OGC format by considering all the CRS components and placing under "the_geom" field of the route table. If this is not evaluated correctly by the system there will no meaningful output on the map. By looking at the OGC format of spatial objects, we can see its complexity for human to understand with bare eyes. This is where GIS techniques helps so in order to make it human understandable and presentable.

Actions	id	communication_protocol	round_trip_delay	signal_strength	latitude	longitude	altitude	bearing	speed	accuracy	total_satellites	satellites_in_fix	the_geom
Edit Delete	1	HSUPA	1854	15	63.41656451	10.39961234	73	234.1	1.74	8	18	13	0101000020E61000008DC116FD99CC2440E88F61FC51B54F40
Edit Delete	2	HSUPA	914	16	63.41655563	10.39957537	73	233.6	2.33	8	18	13	0101000020E6100000CCDF942495CC244047E8E3B151B54F40
Edit Delete	3	HSPA+	128	15	63.41654083	10.39952299	73	235.3	3.31	8	18	14	0101000020E6100000A81900478ECC24403C26BD3551B54F40

3 row(s)

Figure 6.13: A PostgreSQL GUI having Spatial object field "the_geom" in table "route"

6.3 Summary

The creation of the database using PostgreSQL was made enabled to store and handle spatial object by implementing PostGIS functionalities to extend its core capabilities. The PostGIS documentation providing by QGIS Development Team [ug] is referred for installing PostGIS over PostgreSQL. When the data is uploaded directly by the users using the designed NTNU-QGIS webpage, the information contained in the file as a collection of road traffic records updated after a particular interval of time is stored directly in the "route" table. The Postgres GUI can be used to view and alter the table if required. The database is designed in a way that it automatically make use of Spatial Reference System (SRS) for providing accuracy in locating spatial Point objects on the maps. As seen the Open Geospatial Consortium (OGC) format is a complex string of numbers to understand but visualization makes is simpler. The GIS technology, QGIS which we are using in our system development, supports around 2,700 known CRS definitions. The CRS definitions available is defined by *the Institut Geographique National de France (IGNF) and the European Petroleum Survey Group (EPSG)* [qgif].

Design Implementation of Dynamic Visualization of Spatiotemporal Data using PyQGIS

The chapter gives detailed description about implementation of GUI Application using PyQGIS. It is the backbone of the entire system design for flexible and dynamic visualization of spatiotemporal data. The PyQGIS-GUI is developed using python and PyQt4, while to visualize various spatial data on Map view is performed using various QGISmodules. The data is stored on PostgreSQL/PostGIS as described in Section 6.2.2 and so the analysis process is performed on the data by making SQL queries and later presented in the Map-View. The PyQT4 acts as binding for python and QGIS Modules.

7.1 PyQGIS- Application Overview

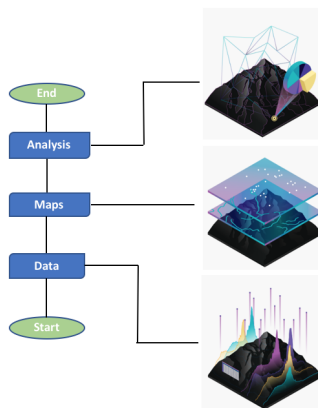


Figure 7.1: This figure will be referenced as Python Standalone Application Overview, some images adopted from [ESR]

7.2 Blueprint of Standalone Script using PyQGIS

In this section, we will see how the various GIS functionalities have been encompassed in the standalone script using PyQGIS (Section 3.2). The main idea behind this is to bring flexibility and scalability for the real-time visualization of various spatial data. Moreover, make analysis and bring meaningful conclusions in practical developments.

7.2.1 Skeleton of PyQGIS Script

The logic behind creating the PyQGIS script is to incorporate both static and dynamicity in the visualization and analysis process of spatiotemporal data as shown in Figure 7.2. This application is the backbone of the entire system architecture which was outlined in Section 3.1 and, additionally it offers direct connectivity with PostgreSQL/PostGIS and NTNU-QGIS web-application.

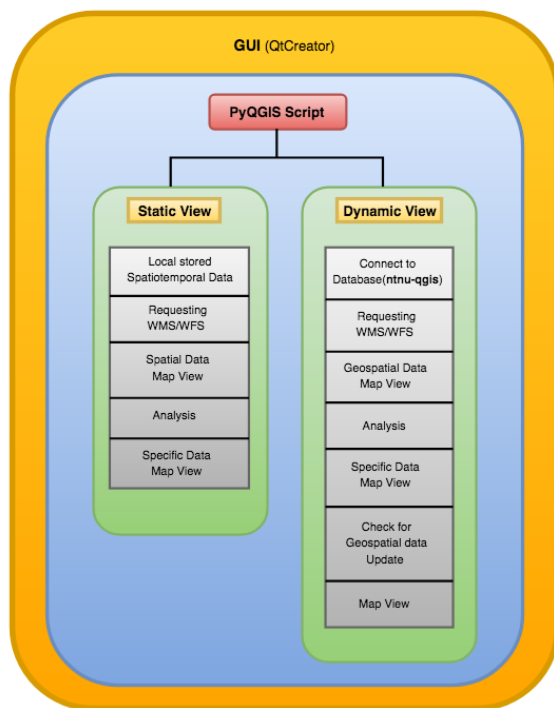


Figure 7.2: This figure shows Logic behind Flexible and Dynamic Visualization of Spatiotemporal-Data in PyQGIS Application

7.2.2 First GUI View of PyQGIS Application

The execution of the application has NTNU-logo as the start-icon followed by the GUI display named as **Visualization of GIS Norge Data** consisting of different features including access to various QGIS features for creating and interacting with the map as shown in Figure 7.3. Hovering over the options will display a short description of the icon's purpose. The following chapter give a working of the entire application and how it was implemented. The application GUI consists of:

- First Top-Level: Menu Bar
 - o Menu Option: **File**
 - o Menu Option: **About**
- Second Middle-Level: ToolBars
 - o Web Option: **Connect to NTNU-QGIS Webpage**
 - o Web Option: **Connect to PostgreSQL/PostGIS Database GUI**
 - o QGIS Option: **Add WMS Map**
 - o QGIS Option: **Create Vector Layer**
 - o QGIS Option: **Where Am I ??**
 - o QGIS Option: **Select Vector Layer**
 - o QGIS Option: **Make a deeper Analysis**
 - o QGIS Option: **Connect to QGIS Server**
- Third Bottom-Level: Buttons
 - o Progress Bar: **Download Map**
 - o Button: **Quit Application Button**

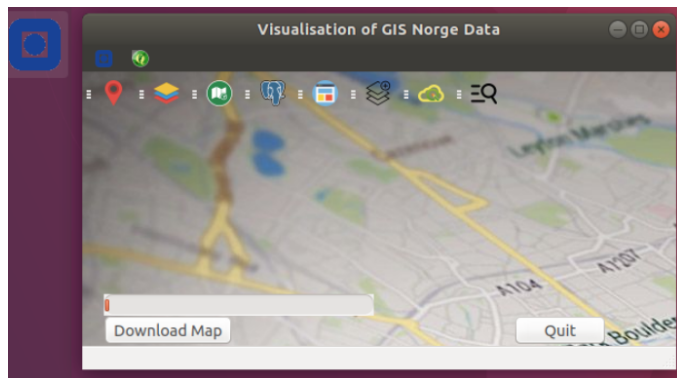


Figure 7.3: This figure shows PyQGIS- GUI View, the background Google map image adopted from *source: <http://miraфра.com/tripsy/2214585-google-maps-wallpaper/>*

62 7. DESIGN IMPLEMENTATION OF DYNAMIC VISUALIZATION OF SPATIOTEMPORAL DATA USING PYQGIS

The Menu option **About**, this feature gives brief explanation about the motivation behind the **Visualization of GIS Norge Data** application development. The illustration of the About feature in Figure 7.4 below.

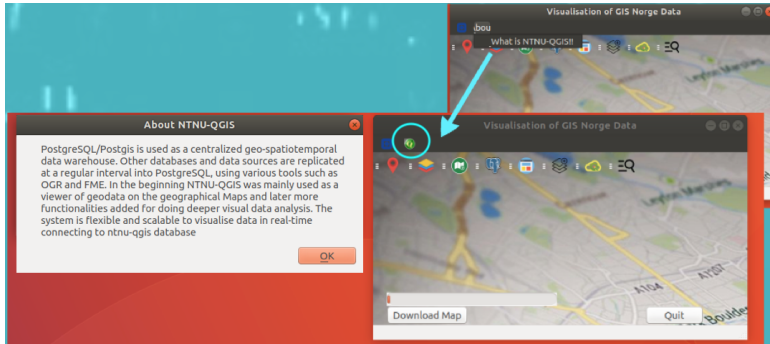


Figure 7.4: This figure shows "About"-Menu Bar Option, providing description of the PyQGIS Application

7.3 Implementation of PyQGIS Application

This PyQGIS application when executed a GUI appears as seen in Figure 7.3. This application is designed to perform both "Static" and "Dynamic" Visualization of spatiotemporal data stored on local machine and/or data captured from "ntnu-qgis" database (Section 5.2.3) respectively.

7.3.1 Static Visualization & Analysis

The "Static" word here implied to the visualization which is not changing automatically. Now when the menu-option **File** is clicked, there is a description stating "Add a CSV file" as seen in Figure 7.5. This when further clicked the local platform's file system window pop-up which provides an ease in navigating to select the spatiotemporal data file in CSV¹ format. Here is the limitation using this option, with respect to this thesis it is build to accept only one defined file format because the Android device (Section 2.4) provides data stored only in CSV file format, but this can be expanding later including more formats to the code with future needs in static data visualization.

¹comma-separated values file, which allow the data to be stored in a tabular format

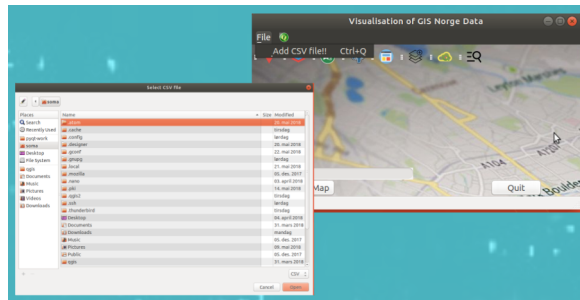


Figure 7.5: This figure shows "File"-Menu Bar Option, providing access to local computer file system to upload Data file

The flow diagram below in Figure 7.6 shows the toolbar features responsible for "Static" visualization and its analysis in yellow rectangular boxes (File and Select Vector Layer). In addition, also shows the further processes. The Static visualization of Spatiotemporal data process are categorized into two flows:

- Flow 1: **Static Data Visualization Process**
- Flow 2: **Analysis: Specific Patterns in Spatiotemporal Data**

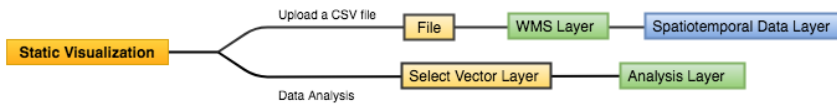


Figure 7.6: This figure shows Flow Diagram for "Static Visualization and its Analysis"

7.3.1.1 Static Data Visualization Process

There are three processes for creating visualization under this flow 1:

- **Spatiotemporal-Data File Selection**

In this project, the CSV file contains various position coordinates along with cellular road traffic information of the car object as shown in Table 7.1. The spatial data information is stored under header *Longitude, Latitude, Altitude* while the other details about the car features encompassing cellular details in presents under header categories as *Communication Protocol, RTD, Signal Strength, Speed, Accuracy, Bearing, Total number of satellites and Satellites in fix*. Any Spatiotemporal CSV file can be used in this static part of the application.

Table 7.1: An Attribute table containing all Car Features saved in a CSV file

Timestamp	Communication Protocol	RTD	Signal Strength	Latitude	Longitude	Altitude	Bearing	Speed	Accuracy	Total number of satellites	Satellites in fix
1,50E+12	EDGE	1414	23	63.41414528432335	10.408184845000505	69.4000015258789	170.46387	4.68	7.5	18	17
1,50E+12	EDGE	788	23	63.414127221331	10.408182833343744	75.4000015258789	161.46057	4.8199997	7.5	18	18
1,50E+12	EDGE	465	23	63.41403330210596	10.408213511109352	78.30000305175781	155.01709	4.08	4.0	19	19
1,50E+12	EDGE	442	23	63.413997050374746	10.40824418887496	75.80000305175781	154.84668	3.53	3.0	21	20
1,50E+12	EDGE	422	23	63.41397232376039	10.408266065642238	75.0	152.72644	2.59	3.0	21	20
1,50E+12	EDGE	413	23	63.413965241674036	10.408285846933722	76.30000305175781	143.94836	2.3999999	3.0	21	19
1,50E+12	EDGE	422	23	63.41393431182951	10.408319877460599	76.19999694824219	131.86889	3.0	3.0	22	21
1,50E+12	EDGE	421	23	63.41391880530864	10.408381400629878	75.4000015258789	118.932495	3.97	3.5	22	18
1,50E+12	EDGE	402	23	63.41390916611999	10.408468656241894	75.5999984741211	97.50366	4.88	3.5	22	20
1,50E+12	EDGE	415	23	63.413911974057555	10.40858130902052	77.30000305175781	84.97925	5.68	3.5	22	21
1,50E+12	EDGE	423	23	63.41391993686557	10.408707289025187	76.0999984741211	80.25513	6.22	3.5	22	21
1,50E+12	EDGE	413	23	63.41393066570163	10.40883595123887	75.0	78.31604	6.89	3.5	22	20
1,50E+12	EDGE	468	23	63.41394441202283	10.408964697271585	73.19999694824219	76.53076	7.2999997	3.5	22	20
1,50E+12	EDGE	427	23	63.413964067585766	10.409112805500627	72.5999984741211	74.30903	7.72	3.5	22	20
1,50E+12	EDGE	413	23	63.413984100334346	10.409260410815477	71.4000015258789	71.81213	7.74	3.5	22	21
1,50E+12	EDGE	421	23	63.41401519719511	10.409403908997774	71.69999694824219	64.989624	7.81	3.5	22	21
1,50E+12	EDGE	421	21	63.414060794748386	10.4095424618572	72.4000015258789	56.30493	7.87	3.5	22	22
1,50E+12	EDGE	489	21	63.414114941842854	10.409663496538997	73.69999694824219	46.801758	7.8599997	3.5	22	20

– **Creation of Basemap Layer (WMS Layer)**

This is the first layer created at the bottom called the basemap layer. The formation of this layer is made using raster data as described Section 2.1.3 and is displayed by requesting WMS services. The logic for developing this layer is same as used by QGIS server Section 4.3.2 and logic implementation using PyQGIS approach is illustrated in later Section 7.3.2.3. This layer acts as a reference to position car objects over the map.

– **Creating Vector Layer from the data file**

This Layer is called a Vector Layer because it contains spatial object in the form of "Point" representing various car features. The layer is formed using various records in the data file as car feature objects and every car object is positioned on the basemap using spatial information (Longitude, Latitude) by using CRS string notation. See a part of logic used in the formation of the spatial features incorporating all on a single layer in Algorithm 7.1.

Algorithm 7.1 Logic to create Vector Layer from CSV file

```
path = "//path/to/file/filename.csv?type=csv&crs=epsg:4726&xField=Longitude&yField=Latitude&spatialIndex=yes&subsetIndex=yes&watchFile=yes"
```

```
vlayer = iface.addVectorLayer(path, "static vector layer", "delimitedtext")
```

– **Result: The Static Visualization**

In the Figure 7.7, the Norwegian Map layer is the WMS-Basemap layer and the dotted (Points) is the vector layer describing various car features captured every 1 second (approximately).

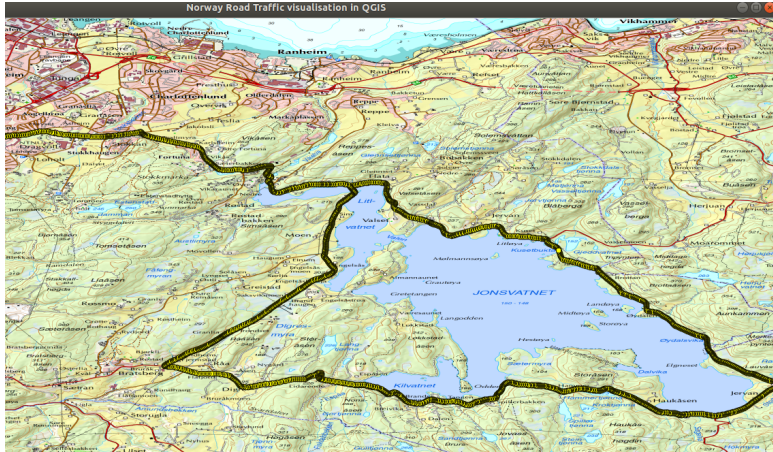


Figure 7.7: This figure shows the Static Visualization of Spatiotemporal data stored in local CSV file

7.3.1.2 Analysis: Specific Patterns in Spatiotemporal Data

Flow 2 is the analysis process of static visualization and various useful patterns captured using toolbar feature developed using PyQGIS. The flow consists of following steps:

– **Toolbar Option: Select Vector Layer**

This option is developed to provide flexibility in selection of any layer which has been created previously by using various data files.

– The Figure 7.8 below illustrates the two QGIS functionality (green Marked circles) present in this option to generate specific patterns while performing analysis over data.

The two functionalities of this option are as follows:

- **Mark 2: Select a Layer:** After clicking on the "Select vector Layer" icon (mark 1) the window will pop-up which will give all the options of the layer already created in the drop down menu seen in the bottom-right side image of the figure below.

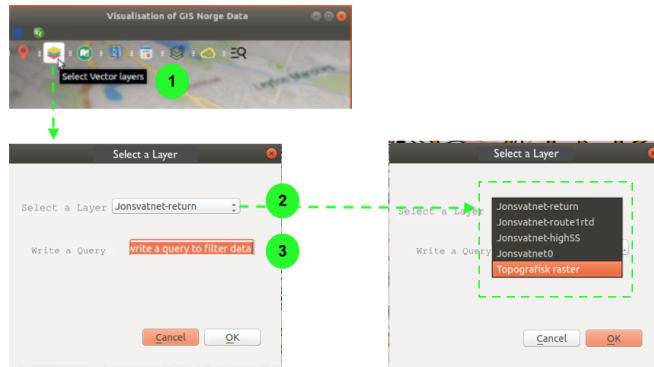


Figure 7.8: This figure shows "Select Vector Layer"- ToolBar Option, providing access to select any layer from all the layers created and make query

- Mark 3: **Write a Query:** After making a selection of the layer, in the text-box (inside orange rectangular space), type a SQL query.
For example:
"Signal Strength" > 20.

– Analysis Result: **The Specific data Map-view**

The Figure 7.9 below shows only the specific data which was queried on the selected layer. We can do any query on the data using the information present inside the file and, the map can be zoomed and panned for detail viewing.

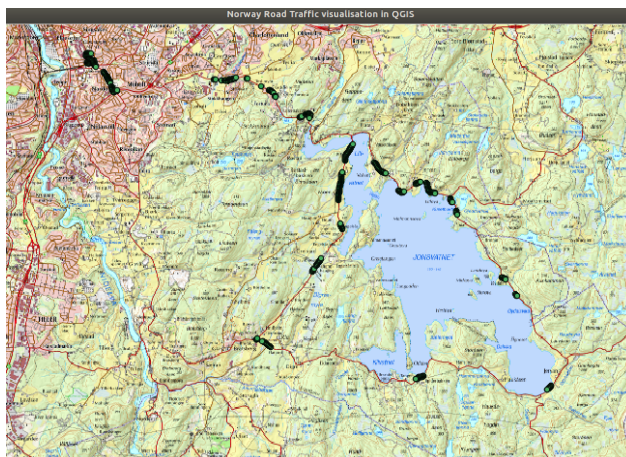


Figure 7.9: This figure shows the zoom out Map-View for Queried data (Signal Strength > 20)

We can select another layer and make query by clicking the toolbar option again and it will display using another color Point object on the same Map-view. This can help during comparison data analysis.

7.3.2 Dynamic Visualization & Analysis

This section describes the development and execution process to achieve a flexible and dynamic visualization of spatiotemporal data using PyQGIS in standalone application. The flow diagram below in Figure 7.10 illustrates the toolbar features (depicted in yellow rectangular boxes) responsible for creating "Dynamic Visualization" and Analysis of the data. The entire process is outlined in three different flows:

- Flow 1: **Uploading & Storing Spatiotemporal Data to the Database**
- Flow 2: **Dynamic Visualization of Data on Map Canvas**
- Flow 3: **Analysis of the Data**

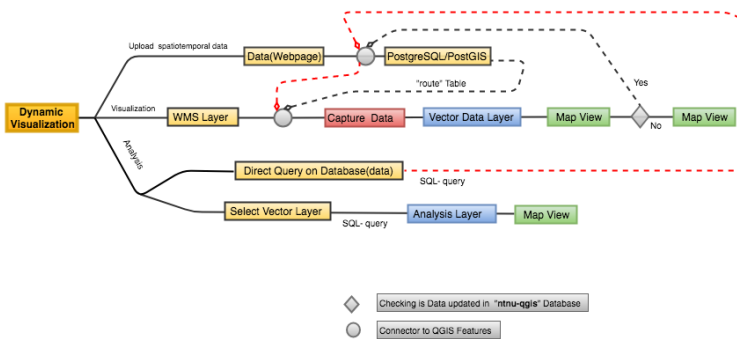


Figure 7.10: This figure shows Flow Diagram for "Dynamics Visualization and its Analysis"

7.3.2.1 Flow 1: Uploading & Storing Spatiotemporal Data to the Database

This section describes the process to gather data and storing it to the database as shown in flow 1 diagram Figure 7.11 below.

The flow consists of the use of two web options embodied in the PyQGIS application-GUI which are responsible for managing and providing the data for visualization:



Figure 7.11: The flow 1: Uploading and Storing Spatiotemporal Data to the Database

– **Process 1: Upload the Spatiotemporal Data**

By clicking on "Connect to NTNU-QGIS Webpage" toolBar option as shown Figure 7.12, the login page pop-up on the browser. By log-in to the page the data of any format is uploaded to the database table and can also be view on the dashboard web page (described in Section 4.2). But this data can also be uploaded from outside and not necessary to use this toolbar option, its only to provide addition support to reach every component in the system architecture present inside one application.

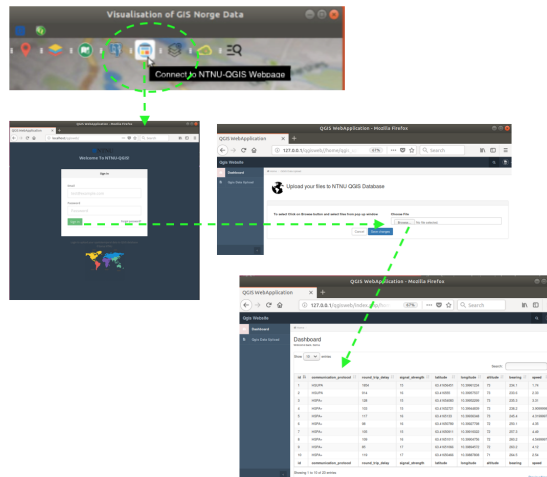


Figure 7.12: This figure shows "Connect to NTNU-QGIS webpage"-ToolBar Icon, providing direct access to NTNU-QGIS login webpage to upload data

– **Process 2: Store the Spatiotemporal Data to "ntnu-qgis" Database**

The data uploaded of any format not restricted to coming from a CSV file is stored into the "ntnu-qgis" database. This data is saved under "route" table and can be seen from PgAdmin GUI where it can be updated or altered. The

PyQGIS- GUI offers the direct access to the PostgreSQL- GUI, by clicking on the "Connect to PostgreSQL database" toolbar option as shown in Figure 7.13. When clicked pgAdmin webpage is open on the browser using Apache web server, where authorized PostgreSQL database user can login and see the data stored.

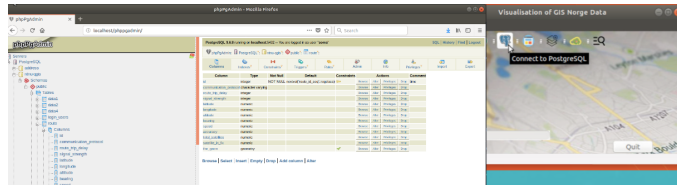


Figure 7.13: This figure shows "Connect to PostgreSQL database"- ToolBar Icon, providing access to (ntnu-qgis) Database

7.3.2.2 Flow 2: Dynamic Visualization of Data on Map Canvas

This section describes the development process of creating dynamic layer according the data coming directly from the database and placing the position of every car object accurately over the basemap layer. As seen in the flow diagram Figure 7.14 below, the execution process includes three stages:

- Stage 1: **Creating WMS Layer (Basemap)**
- Stage 2: **Creating Vector Layer from Database data**
- Stage 3: **Checking for Data Updates**

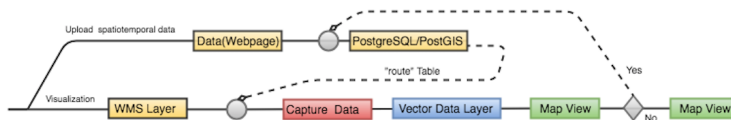


Figure 7.14: The flow 2: Dynamic Visualization of spatiotemporal database data on Map-view process

Let us see in the following sections how various stages are implemented to achieve flexible and dynamic system to visualize the spatiotemporal data being stored into the database table by flow 1.

7.3.2.3 Stage 1: Creating WMS Layer (Basemap)

In the previous Chapter 3 from the blueprint of system architecture Section 3.1, we see the PyQGIS application is requesting WMS/WFS Services from the the cloud². We used WMS services in our system implementation for creating basemap Layer. The Web Map Services (WMS), *is a standard protocol which is developed in 1999 by the Open Geospatial Consortium (OGC) for providing georeferenced³ map images which are served by various map servers over the internet as seen in Figure 7.15 [HKKM14]. The images served is in the form of raster data which a computer depicts as cells (pixels) consider referring Section 2.1.3 [dLBD02].*

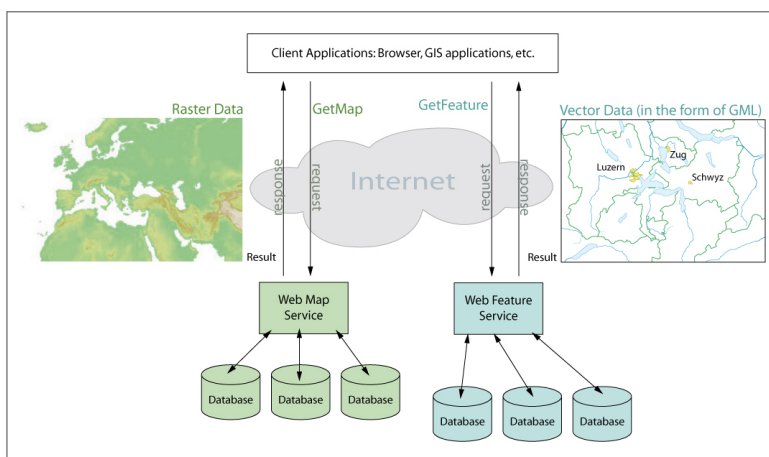


Figure 7.15: The figure shows the WMS/WFS basic architecture for serving map over the cloud adopted from [fSHE]

When the "Add WMS Map" toolbar option is clicked a **Norge WMS Raster map using QGIS** window pop-up and which displays a zoom-out image view of the Norwegian Geographical map as shown in Figure 7.16, which can be zoomed-in and panned for detail viewing. This image is the bottom-most layer (referred as Basemap Layer) which is created from the raster Data collected by requesting WMS services over the Internet. We have seen in previous section the two ways of requesting WMS services (Section 4.3.2). For this project the WMS service is requested from "GEONORGE" web server [GEOB] using python script to serve Norway map as we are dealing with data gathered from "Trondheim" we do not need the entire world map which will in a way slow the performance. But there is no limitation is choosing WMS services, this can be replaced with other WMS services offering different geographical map images over the Internet by various web map servers.

²is also phrased as "cloud" and used as a metaphor for the "Internet"

³<https://en.wikipedia.org/wiki/Georeferencing>

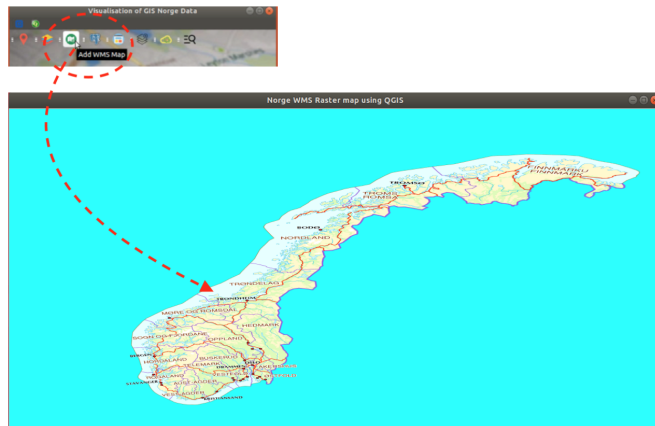


Figure 7.16: This figure shows "Add WMS Map"-ToolBar Icon, to create and display Norge BaseMap Layer

The logic used for capturing raster data is illustrated in Algorithm 7.2 contain three steps to create and visualize the Norway map. First, we request the Web server using "GetMap" to serve the map image by providing its content in the form of raw raster data. The raw raster data captured is then converted by using various QGIS module to generate a raster layer in the form of image (collection of pixels) and lastly the layer is added to the "Map Registry"⁴ to display on the map canvas with CRSas EPSG: 4326 to appear accurately on the 2D map surface. The zoom-out map image can be panned or zoom-inside/zoom-out.

7.3.2.4 Stage 2: Creating Vector Layer from Database data

When the spatiotemporal data is uploaded from webpage and stored into the database as previously described Section 7.3.2.1. The database layer is created in the form of vector layer as the information stored in the table of "ntnu-qgis" database contains the road traffic record which is a spatial object. The records are represented as "Points" car object which is referred as vector point together forming a vector layer. The Layer is created by using the "Create vector layer" toolbar option present in the PyQGIS- GUI as seen in Figure 7.17.

⁴is the repository that stores all the vector/raster layer in the organized one over the other as displayed

Algorithm 7.2 Requesting Norge WMS services and creating Basemap Raster Layer
Requesting WMS serves to serve Norge Map- Raster Data

```
urlWithParameters = "contextualWMSLegend=0&crs=EPSG:4326&
dpiMode=7&featureCount=10&format=image/png&layers=
topografiskraster&styles=&GetMap&
url=https://openwms.statkart.no/skwms1/wms.toporaster3?
version%3D1.3.0%26"
```

Creating the Bottom-most Raster-map layer

```
raster_layer = QgsRasterLayer(urlWithParameters, "WMS Norge",
                              "wms")
```

Checking if the Raster Layer developed & adding to Map Registry

```
if not raster_layer.isValid():
    raise IOError, "Failed to load WMS raster Layer"
    QMessageBox.warning(None, "Invalid WMS", "WMS raster layer
                        failed to load!")
else:
    QgsMapLayerRegistry.instance().addMapLayer(raster_layer)
```

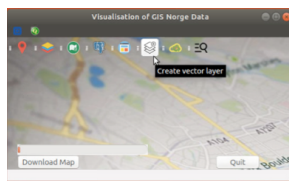


Figure 7.17: This figure shows "Create vector layer"-Toolbar Option, creating vector layer from data stored in "ntnu-qgis" Database

When this icon is clicked in perform some functions coded in standalone application. Firstly, it makes the connection with the "ntnu-qgis" database and check the connectivity is stable. Secondly, it captures the data from the database which is in the form of vector format. By using some QGIS modules, the vector layer is formed and as the vector layer contains various car features represented in the spatial form as "Point" on the layer. As we know these spatial forms are complex structure to understand by looking in the vector data (Open Geospatial Consortium format) so they represented in as visual Points to make understandable presentation. The location of each car feature (as "Point" spatial object) is decided by using the "the_geom" column from the "route" table which was geo-spatially converted using PostGIS. The conversion is

important to make the spatial car features placed accurately on the vector layer.

After creating the vector layer it is then ready to add to the Map Registry to place in an orderly format to display on the map canvas as shown in Figure 7.18. The layer in the Map Registry are placed one over the other as they are formed just like in a stack, for instance the basemap is created first so it is the first layer (bottom layer) then the vector layer is the second layer created (top layer) and so on.

Furthermore, as this vector layer sits on the top of the basemap layer created previously using WMS service, the basemap acts as a geo-reference for understanding the vector data layer placing on the map. If the CRS (ESPG : 4326) for both layers is not the same then each vector points will be projected somewhere else on the globe. The basic logic for making connection with the PostgreSQL database and the dynamic visualization of the vector data is briefed in Algorithm 7.3.



Figure 7.18: This figure shows Dynamic Vector Layer Visualization on the Map Canvas

Algorithm 7.3 Basic logic to create database Vector layer

```

database_uri = QgsDataSourceURI()
database_uri.setConnection("localhost", "5432", "ntnu-qgis",
                           "soma", "****")
database_uri.setDataSource("public", "route", "the_geom")
.
.
.
QgsMapLayerRegistry.instance().addMapLayer(QgsVectorLayer
      (database_uri.uri(True), "database layer", "postgres"))

```

7.3.2.5 Stage 3: Checking for Data Updates

This part play a very crucial role in the development of the flexible and dynamic visualization process. After achieving the first database data visualization, it is possible that data might be altered or new data might be uploaded simultaneously. So in order to have real-time visualization we maintaining a continuous connectivity with the "ntnu-qgis" database which means that the standalone scripts is build in a way that in every short intervals, is checks the connectivity with the database. In addition, it also compares the old data with the data present in the "route" table at that time. If there is any update in the data then the new vector is formed and replace the old vector layer displayed on the map automatically. This is shown in above flow diagram Figure 7.14.

7.3.2.6 Flow 3: Analysis of the Data

The process of doing analysis is very important to find hidden patterns and make future prediction for better development of any system. The analysis of the data which is stored in the database table can be analyzed in using two different approaches. The two approaches are as follows:

– **Analysis 1: Select Vector Layer**(Toolbar Option)

The process of using this feature is same as described in the previous Section 7.3.1.2. The Figure 7.19 shows the flow of this analysis process.

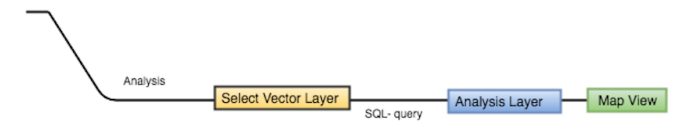


Figure 7.19: This figure shows the flow diagram for (database) Data Analysis using "Select vector Layer"-Toolbar Option

The only difference here is that the database vector layer will also be displayed along with the queried data layer which is appearing on the top with new color as seen in Figure 7.20. And this is happening because the data is being fetch continuous from the database and making simultaneously visualization.

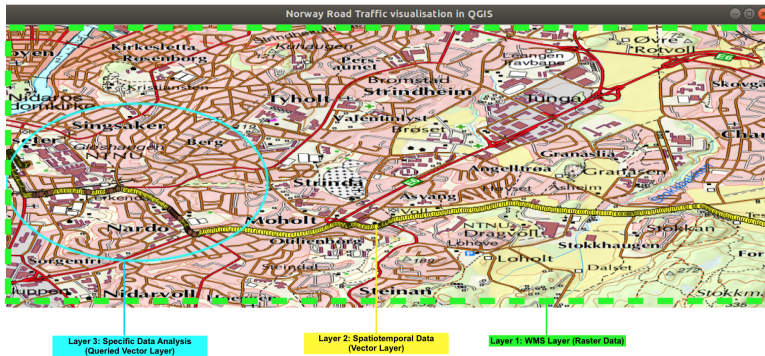


Figure 7.20: This figure shows the Analysis Result obtained using "Select Vector Layer"-Toolbar Option on Database Vector-Layer

– **Analysis 2: Make a Deeper Analysis** (Toolbar Option)

In this analysis process, the data is first queried directly on the data stored in "route" table and then layer is performed according to the data captured. The Figure 7.21 shows a flow diagram for making a direct query onto the database data.

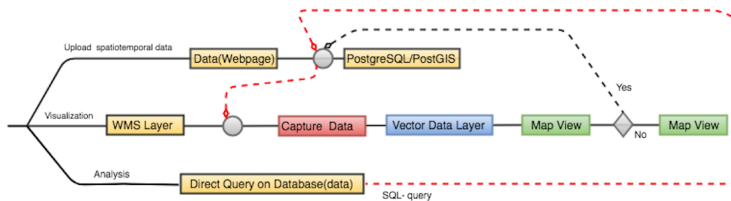


Figure 7.21: This figure shows flow of analysis process directly on the database data

The process of this analysis is demonstrated in three green markings. When the "Make a deeper Analysis" toolbar is clicked (mark 1) as shown in Figure 7.22, immediately the "Vector Layer Analysis!" dialog box pop-up and as from the figure (mark 2) which has two option either yes or no. If option "no" is selected no analysis is performed but if clicked "yes", the " Make a deeper analysis" Query Box appears. In the text box (mark 3) write a SQL query as same performed in Section 7.3.1.2. Here the advantage is that the data can be referred by seeing on the dashboard web page Section 4.2.1.4 to make meaningful query

and obtaining visual patterns.

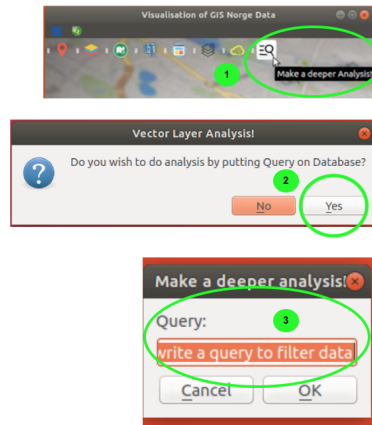


Figure 7.22: This figure shows "Make a deeper Analysis"- ToolBar Option, and steps to make query

The benefit of using this process for data analysis is, it enhances performance in making faster data query and quicker visualization on the map canvas than performed by above Analysis 1 process. As the data is directly queried from the database table which result in gathering specific amount of data before visualizing on the map view. So instead of making first all the thousand(s) spatial objects appear over the map, now it has to locate few among them and create specific patterns. In addition, while this process is executing and suddenly data in the table is changed then a notification will appear notifying the change in data and after few seconds of waiting it will refresh the screen and ask to make new query. The Figure 7.23 shows the result displayed the specific queried data in the dynamic visualization process.

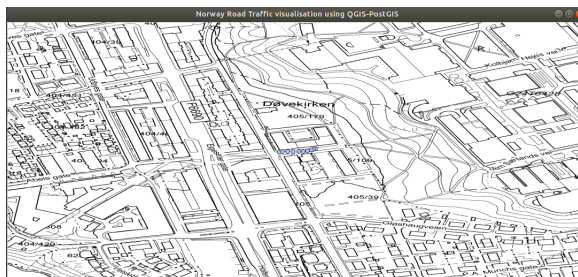


Figure 7.23: This figure shows "Make a deeper Analysis"- ToolBar Option, Analysis result

There are various other features available in the PyQGIS- GUI. Which includes:

7.3.2.7 Find out Position Coordinate of random Car Feature

This QGIS functionality is very useful in order to find out the position coordinate of the random selected car feature. The toolbar option "where am I" when clicked as shown in figure Figure 7.24, a cross arrow mouse pointer is generated that hover over the map [She14].

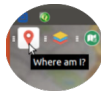


Figure 7.24: This figure shows "Where AM I??"-Toolbar Option, to capture specific Position Coordinate(Longitude,Latitude) by click of a Mouse

When the cross arrow pointer is clicked on any car feature or any location over the map, the window box "where am I" is pop-up which displays the Longitude and Latitude of that selected location as shown in Figure 7.25. This toolbar option can be used anytime during both static and dynamic data visualization process.

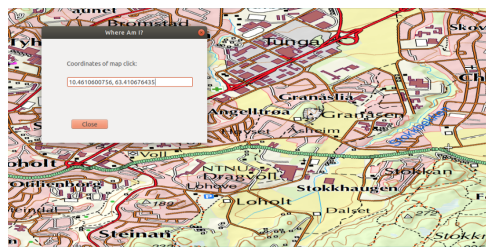


Figure 7.25: This figure shows Map view capturing specific car feature Position Coordinates (Longitude,Latitude)

7.3.2.8 Creating Connection with the QGIS Server

This toolbar option "Connect to QGIS Sever", it is the additional QGIS functionality present in PyQGIS - GUI application. When clicked, the QGIS server is accessible through opening on the browser using Apache web server. The layers generated during static visualization process can be seen using QGIS server from other part of the globe. But here the data layers will be displayed as a fixed image and cannot be changed.

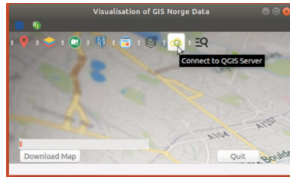


Figure 7.26: This figure shows "Connect to QGIS Server"- ToolBar Option, providing access to QGIS Server

7.3.2.9 Downloading Map- Progress bar

This "Download Map" progress option button as shown in Figure 7.27 is very helpful in case we want to save any of the visualized images in the form of PDF version for later examination. Like during the case of making query directly on the database table and when the notification received about data updated, this can be useful in saving the pattern to compare with new patterns. The file is directly saved by default in the download folder of the local platform. The basic logic is defined in Algorithm7.4 [SITa]. Basically when this button (Download Map) is pressed it captures the current view on the map canvas.

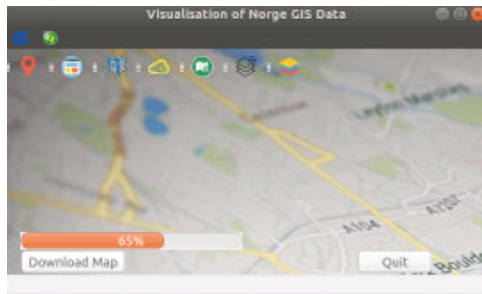


Figure 7.27: This figure shows "Download Map"-Progress Bar Option, creating PDF of layers Displayed on Map-Canvas

7.3.2.10 Quit Button

This button when clicked closes the application at any point of time and no matter of any process is running. This when executed everything is shut.

Algorithm 7.4 Basic logic to print the Map Canvas in PDF

```

#caoturing the map canvas
map_render = QgsMapRenderer()
layer_list = [layer.QgsMapLayerRegistry.instance().mapLayers()]
map_render.setLayerSet(layer_list)
rect = QgsRect(map_render.fullExtent())
rect.scale(1.1)
map_render.setExtent(rect)
.
.
.

#Set painter
print = QPainter()
print.setOutputFormat(QPrinter.PdfFormat)
print.setOutputFileName("ntnu_qgis_Map_canvas.pdf")
print.setPaperSize(QSizeF(c.paperWidth(), c.paperHeight()),
QPrinter.Millimeter)
print.setFullPage(True)
print.setColorMode(QPrinter.Color)
print.setResolution(c.printResolution())
.
.
.
map_render.render(print)
pdf_print = QPainter(print)
paperRect_mm_size = print.pageRect(QPrinter.Millimeter)
paper_RectPixel = print.pageRect(QPrinter.DevicePixel)
c.render(pdf_print, paper_RectPixel, paperRect_mm_size)
pdf_print.end()

```

This chapter gave a detailed implementation process description on the development of PyQGIS application and its functionalities. The PyQGIS application source code is provided in appendix .3. The next Chapter 8 is the summarization of this entire thesis process.

Chapter 8

Discussion and Conclusion

8.1 Discussion

Setting up the dynamic visualization of spatiotemporal data was the main objective of this thesis work. The implementation of the main dynamic system application has been described in the previous Chapter 7 which embodies other system components as demonstrated in the blueprint of system architecture diagram Section 3.1.

The complete project comprises of three most important developments in creating a flexible and dynamic visualization for Norwegian road traffic data. The first was designing and implementing the system which will interact with external devices which are responsible to gather the data. For this purpose, the web application is created which is a collection of three web pages performing different operations. The next important component of the system was to create something which can store the captured data for visualization. But this data is not a normal form of data which can be understood and interpreted by glancing. This is location-dependent data and in the form of spatial data. These data are basically complex structure storing various shaped like a point, line, etc. In order to handle spatial datasets, we choose to work with PostgreSQL for designing a separate database. These data contain records and each record implies the information about the spatial object which in this thesis is a car. As records collected after certain time interval so it is a discrete data which in the spatial domain is called as a vector data. We created a spatial database to handle these complex values by extending the core capabilities of the PostgreSQL using PostGIS which is not commonly offered by other DBMS.

We have designed the uploading web page in a manner that when the data reaches the database, the position coordinates automatically get converted into the form which is recognized by QGIS to create a visual object for human understanding. The position coordinates of car feature which are latitude and longitude are in decimal units and in order to make it QGIS language it has to be converted into OGC format which is a string of number formed together using coordinated, correct projection

information which is called CRS conversion. The conversion is necessary because the data captured is from the 3D world and we need to display on 2D map by locating correctly on flat surface maps and this is enabled using CRS conversion algorithms.

By this time, we have gathered the data and stored in a spatial format, and now it is ready to get visualized. There are many GIS tools available in the market today, out of them some are developed for free use and some are commercially bounded. We used QGIS and the main benefit of using this as it is freely available and it provides easy access to use its repository in creating more flexible and intelligent visualizations. Using QGIS libraries we created a python application which is the backbone of the entire system, which gives greater speed and performance. In addition, the `qgis.core` and `qgis.gui` libraries help to produce excellent spatial data visualization image according to our required purpose.

The map canvas used to visualize the vector is developed to produce a dynamic graphical representation of the data. Which means the visualization process automatically changes with the data supplied. This is also a way of tracking various objects.

An analysis is the main process for exploration of data and gathers meaningful pattern which is hidden. Of course, the analyzing process influence on the data visualization. The python application is developed to make SQL queries directly from the database which in a way increases the performance of data visualization speed. We have already created light-weight spatial indexes in the database which makes querying on the data faster. The patterns obtained as results gives decision makers to makes useful predictions for future developments in the cellular network management.

8.2 Limitation and Challenges in Current Work

By considering the complexity of the spatial structures, we managed to create a dynamic system for visualization purposes in the spatial domain. Some challenges faced during environmental setup are previously described in Chapter 3 but one major challenge that affected the performance of entire system design was overlaying the layers one over the other. The Qgis modules are not flexible to any modification. Every defined visualization modules have to be used in a certain way for creating a layer from the dataset The vector layer in comparison to the raster layer was quick in the formation. So during the entire testing of the application, this was the major challenge to control the display of various layer. The raster layer needs continuous connectivity with the specified map server for its formation. And sometimes, the connectivity is slow which automatically affects the process of base map layer generation. While on the other hand, the vector layer is formed using datasets from the database. Here also it makes continuous connectivity but comparing to the size of

both data, the size of objects is smaller than gathering cells of the raster image. This creates spatial objects sometimes before collecting all the pixels for forming raster image even though the code is arranged systematically according to the displaying order for the layers. It was overcome by producing inbuilt delay between both layer formation. Once the stability and overlaying of the layer are achieved correctly, the analysis process could be performed with an ease.

8.3 Conclusion

In this project thesis, with the development of the dynamic system, we have shown that it is possible to visualize spatiotemporal data flexibly in real-time and which can also be examined from anywhere. The achieved system is capable to store any form of data and convert it to specific format automatically. Along with that the development of NTNU-QGIS web API will provide access to various devices to simultaneous collect and store data in the database to visualize at the same time. The entire system is scalable and is also detachable to integrate with other applications in the future. The various spatial patterns resulted in analysis process by making queries aid in giving a deeper understanding of the road traffic system which in a way provide answers to some questions related to improving the cellular network in road traffic.

References

- [Alt] Mark Altaweel. *The PostGIS*. <https://www.gislounge.com/what-is-postgis/> [Accessed: 15-05-18].
- [BCG14] Sudipto Banerjee, Bradley P Carlin, and Alan E Gelfand. *Hierarchical modeling and analysis for spatial data*. CRC press, 2014.
- [Bou] Boundless. *Introduction to PostGIS*. <http://workshops.boundlessgeo.com/postgis-intro/#> [Accessed: 15-05-18].
- [dLBD02] Jeff de La Beaujardière and Allan Doyle. Web map service implementation specification, version 1.1. 0. *Open Geospatial Consortium Inc*, 2002.
- [dvi] dvisualization. *Traffic Data Visualization*. <https://dvisualization.wordpress.com/2017/03/27/hello-world/> [Accessed: 19-06-18].
- [en.] en.wikipedia.org. *Spatiotemporal database*. https://en.wikipedia.org/wiki/Spatiotemporal_database [Accessed: 16-05-18].
- [ESR] ESRI. *What is GIS?* <https://www.esri.com/en-us/what-is-gis/overview> [Accessed: 19-05-18].
- [ew] en wiki. *Model-view-controller*. <https://en.wikipedia.org/wiki/Model-view-controller> [Accessed: 30-05-18].
- [fSHE] Cartography for Swiss Higher Education. *OGC, WMS and WFS*. http://www.e-cartouche.ch/content_reg/cartouche/webservice/en/html/webservice_LSummary.html [Accessed: 25-05-18].
- [Geoa] National Geographic. *GIS (geographic information system)*. <https://www.nationalgeographic.org/encyclopedia/geographic-information-system-gis/> [Accessed: 19-05-18].
- [GEOb] GEONORGE. *Toporaster 3*. <https://kartkatalog.geonorge.no/metadata/kartverket/toporaster-3/13e15833-8d94-4b89-a53d-eb8da289677b> [Accessed: 25-05-18].
- [Glo] David Gloag. *Geovisualization: Tools Techniques*. <https://study.com/academy/lesson/geovisualization-tools-techniques.html> [Accessed: 18-05-18].

- [gnu] gnu.org. *GNU General Public License*. <https://www.gnu.org/licenses/old-licenses/gpl-2.0.txt> [Accessed: 19-05-18].
- [Gro] PTV Group. *Fusion and visualization of map data and historical traffic data*. <http://vision-traffic.ptvgroup.com/nl/products/ptv-visum-data-analytics/use-cases/fusion-and-visualization-of-map-data-and-historical-traffic-data/> [Accessed: 19-06-18].
- [GS05] Ralf Hartmut Güting and Markus Schneider. *Moving objects databases*. Elsevier, 2005.
- [Hen] Max Henderson. *Traffic Analytics*. <http://maxwellhenderson.com/traffic.html> [Accessed: 19-06-18].
- [HH03] Robert Haining and Robert P Haining. *Spatial data analysis: theory and practice*. Cambridge University Press, 2003.
- [HKKM14] Andreas Hackeloeer, Klaas Klasing, Jukka M Krisp, and Liqiu Meng. Georeferencing: a review of methods and applications. *Annals of GIS*, 20(1):61–69, 2014.
- [JOU] LINUX JOURNAL. *Getting Started With Quantum GIS*. <https://www.linuxjournal.com/content/getting-started-quantum-gis> [Accessed: 20-05-18].
- [Liz] LizMap. *Lizmap 3.0*. <https://docs.3liz.com/en/introduction.html> [Accessed: 28-05-18].
- [Mat] MathWorks. *What Is Geospatial Data?* <https://www.mathworks.com/help/map/what-is-geospatial-data.html> [Accessed: 18-05-18].
- [Mic] Microsoft|Developer. *Introduction to REST and .net Web API*. <https://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/> [Accessed: 31-05-18].
- [NEW] GPS BUSINESS NEWS. *HERE Issues Open Source Specs to Share Road Data From Car Sensors*. https://www.gpsbusinessnews.com/HERE-Issues-Open-Source-Specs-to-Share-Road-Data-From-Car-Sensors_a5539.html [Accessed: 19-02-18].
- [oCa] Earth Lab-University of Colorado. *Coordinate Reference System and Spatial Projection*. <https://www.earthdatascience.org/courses/earth-analytics/spatial-data-r/geographic-vs-projected-coordinate-reference-systems-UTM/> [Accessed: 16-05-18].
- [oCb] Earth Lab-University of Colorado. *GIS in R: Understand EPSG, WKT and other CRS definition styles*. <https://www.earthdatascience.org/courses/earth-analytics/spatial-data-r/understand-epsg-wkt-and-other-crs-definition-file-types/> [Accessed: 16-05-18].

- [oCc] Earth Lab-University of Colorado. *GIS With R: Projected vs Geographic Coordinate Reference Systems*. <https://www.earthdatascience.org/courses/earth-analytics/spatial-data-r/intro-to-coordinate-reference-systems/> [Accessed: 16-05-18].
- [opa] QGIS official page. *QGIS as OGC Data Server*. https://docs.qgis.org/2.8/en/docs/user_manual/working_with_ogc/ogc_server_support.html [Accessed: 20-05-18].
- [opb] QGIS official page. *QGIS Browser*. https://docs.qgis.org/2.8/en/docs/user_manual/qgis_browser/qgis_browser.html [Accessed: 20-05-18].
- [opc] QGIS official page. *QGIS GUI*. https://docs.qgis.org/2.18/en/docs/user_manual/introduction/qgis_gui.html [Accessed: 20-05-18].
- [ore] oreilly.com. *What is MVC?* <https://www.safaribooksonline.com/library/view/laravel-design-patterns/9781783287987/ch01s02.html> [Accessed: 28-05-18].
- [os] QGIS official site. *Lesson: Install QGIS Server*. https://docs.qgis.org/2.18/en/docs/training_manual/qgis_server/install.html [Accessed: 28-05-18].
- [ow] QGIS official website. *Foreword*. https://docs.qgis.org/2.18/en/docs/user_manual/preamble/foreword.html [Accessed: 19-05-18].
- [posa] postgres.net. *The PostGIS Development Group*. <http://postgis.net/docs/manual-2.4/> [Accessed: 15-05-18].
- [posb] postgresql.org. *WHAT IS POSTGRESQL?* <https://www.postgresql.org/about/> [Accessed: 15-05-18].
- [Qgia] Qgis.org. *Documentation QGIS 2.18*. https://docs.qgis.org/testing/en/docs/training_manual/database_concepts/data_model.html [Accessed: 15-05-18].
- [Qgib] Qgis.org. *Documentation QGIS 2.18: Database Concepts with PostgreSQL*. https://docs.qgis.org/testing/en/docs/training_manual/database_concepts/index.html [Accessed: 15-05-18].
- [qgic] qgis.org. *PostGIS setup*. https://docs.qgis.org/testing/en/docs/training_manual/spatial_databases/spatial_functions.html [Accessed: 15-05-18].
- [qgid] qgis.org. *Simple Feature Model*. https://docs.qgis.org/testing/en/docs/training_manual/spatial_databases/simple_feature_model.html [Accessed: 17-05-18].
- [qgie] qgis.org. *Spatial Database Concepts with PostGIS*. https://docs.qgis.org/testing/en/docs/training_manual/spatial_databases/index.html [Accessed: 15-05-18].
- [qgif] qgis.org. *Working with Projections*. https://docs.qgis.org/2.8/en/docs/user_manual/working_with_projections/working_with_projections.html [Accessed: 17-05-18].

- [Quo] Quora. *What is the difference between spatial-temporal data with other type of data?* <https://www.quora.com/What-is-the-difference-between-spatial-temporal-data-with-other-type-of-data> [Accessed: 18-05-18].
- [She14] Gary Sherman. *The PyQGIS Programmer's Guide*. Locate Press, 2014.
- [SITa] QGIS OFFICIAL SITE. *Map Rendering and Printing*. https://docs.qgis.org/2.14/en/docs/pyqgis_developer_cookbook/composer.html [Accessed: 25-05-18].
- [sitb] QGIS Official site. *QGIS as OGC Data Server*. https://docs.qgis.org/2.8/en/docs/user_manual/working_with_ogc/ogc_server_support.html [Accessed: 28-05-18].
- [site] QGIS Official site. *QGIS Installers*. <https://qgis.org/en/site/forusers/alldownloads.html#linux> [Accessed: 28-05-18].
- [Sou] Sourceforge. *Quantum GIS*. http://freshmeat.sourceforge.net/projects/qgis/?branch_id=31471&release_id=274022 [Accessed: 20-05-18].
- [Spr] K. S. Rajan SpringerLink, Sarthak Agarwal. *Performance analysis of MongoDB versus PostGIS/PostgreSQL databases for line intersection and point containment spatial queries*. <https://link.springer.com/article/10.1007/s41324-016-0059-1> [Accessed: 15-05-18].
- [stu] study.com. *Geospatial Data: Definition Example*. <https://study.com/academy/lesson/geospatial-data-definition-example.html> [Accessed: 19-05-18].
- [Teca] TechAdmin. *How to Install Python 2.7.14 on Ubuntu LinuxMint*. <https://tecadmin.net/install-python-2-7-on-ubuntu-and-linuxmint/> [Accessed: 20-05-18].
- [Tech] TechTarget. *database management system (DBMS)*. <https://searchsqlserver.techtarget.com/definition/database-management-system> [Accessed: 19-06-18].
- [THE] THEKINIGROUP. *Importance of data visualization*. <https://thekinigroup.com/importance-data-visualization/> [Accessed: 19-02-18].
- [ug] Boston GIS user group. *PostGIS ver. 2.1 Geometry/Geography Quick Guide - Cheatsheet*. http://www.bostongis.com/postgis_quickguide.bqg [Accessed: 15-05-18].
- [Uni] Humboldt State University. *Raster To Vector*. http://gsp.humboldt.edu/OLM_2015/Lessons/GIS/08%20Rasters/RasterToVector.html [Accessed: 21-05-18].
- [web] webapplicationBodvoc. *An Overview of Internet connection*. <https://bodvoc.wordpress.com/2010/07/02/an-overview-of-a-web-server/> [Accessed: 28-05-18].

Appendix

Source Code: Dynamic Visualization System

.1 Source Code: Static Visualization

```
1 # adding CSV Layer
2
3 from PyQt4.QtCore import *
4 from PyQt4.QtGui import *
5 from qgis.core import *
6 from qgis.utils import *
7 from qgis.gui import *
8 import numpy
9 import requests
10 import sys
11 import os
12 import psycopg2
13 import time
14
15
16 # configure your the prefix path
17 QgsApplication.setPrefixPath("/usr", True)
18
19 #creating a reference to the QgsApplication
20 csv_qgs= QgsApplication([], True)
21
22 # loading the data provider of QGIS and layer registry
23 QgsApplication.initQgis()
24
25 """ creating Map canvas """
26 canvas = QgsMapCanvas()
27 canvas.setCanvasColor(Qt.green)
28 canvas.enableAntiAliasing(True)
29 canvas.setWindowTitle("Norway Road Traffic visualisation in QGIS")
30
31
32 """ creating a vector layer """
33 filename_path = QFileDialog.getOpenFileName(None, "Select CSV file", os
    .getenv("HOME"), "CSV (*.csv)")
34 if filename_path:
```

```

35     print("The file csv path is " , filename_path)
36     print(filename_path)
37 else :
38     # Give a warning QMessageBox if file does not exists
39     QMessageBox.warning(None, "Invalid File", " This CSV file is not
        valid.")
40
41 # the path
42 csv_path = filename_path
43 csv_filter = "?type=csv&xField=Longitude&yField=Latitude&spatialIndex=
        yes&subsetIndex=yes&watchFile=yes"
44 full_path = "file://" + csv_path + csv_filter
45
46 # make the QGIS vector layer
47 csv_layer = QgsVectorLayer(full_path , " Vector Layer 1", "delimitedtext"
        )
48
49 # check is layer valid
50 if not csv_layer.isValid():
51     raise IOError, "Failed to open the Vector Layer"
52     QMessageBox.warning(None, "Invalid Layer", " This QGIS Vector layer
        is not valid.")
53
54 # change the symbol in the csv layer
55 symbol = QgsMarkerSymbolV2.createSimple({"name": "circle", "color": "
        yellow"})
56 csv_layer.rendererV2().setSymbol(symbol)
57 QgsMapLayerRegistry.instance().addMapLayer(csv_layer)
58
59 layers = QgsMapLayerRegistry.instance().mapLayers()
60 map_canvas_layer_list = [QgsMapCanvasLayer(i) for i in layers.values()]
61
62 canvas.setExtent(csv_layer.extent())
63 canvas.setLayerSet(map_canvas_layer_list)
64
65 canvas.show()
66
67 csv_qgs.exec_()

```

.2 Source Code: Dynamic Visualization

```

1 # creating layer from taking GIS data from
   Toolbar_Connect_to_PostgreSQL
2
3 from PyQt4.QtCore import *
4 from PyQt4.QtGui import *
5 from qgis.core import *
6 from qgis.utils import *
7 from qgis.gui import *
8 import numpy
9 import requests
10 import sys
11 import os
12 import psycopg2
13 import time
14
15
16 # configure your the prefix path
17 QgsApplication.setPrefixPath("/usr", True)
18 #creating a reference to the QgsApplication
19 db_qgs= QgsApplication([], True)
20 # loading the data provider of QGIS and layer registry
21 QgsApplication.initQgis()
22
23 """ creating Map canvas """
24 db_canvas = QgsMapCanvas()
25 db_canvas.setCanvasColor(Qt.white)
26 db_canvas.enableAntiAliasing(True)
27 db_canvas.setWindowTitle("Norway Road Traffic visualisation using QGIS-
   PostGIS")
28
29 urlWithParameters = "contextualWMSLegend=0&&crs=EPSG:4326&dpiMode=7&
   featureCount=10&format=image/png&layers=topografiskraster&styles=&
   url=https://openwms.statkart.no/skwwms1/wms.toporaster3?version%3D1
   .3.0%26"
30 wms_raster_layer = QgsRasterLayer(urlWithParameters, "WMS Norge", "wms"
   )
31
32 if not wms_raster_layer.isValid():
33     raise IOError, "Failed to load WMS raster Layer"
34     QMessageBox.warning(None, "Invalid WMS", "WMS raster layer failed to
   load!")
35
36 QgsMapLayerRegistry.instance().addMapLayer(wms_raster_layer)
37
38
39 time.sleep(4)
40
41
42 """ creating a vector layer using PostGIS """
43
44 # making connection with ntnu-qgis database

```



```

45
46 uri = QgsDataSourceURI()
47 #set host name, port, database name, username, password
48 uri.setConnection("localhost", "5432", "ntnu-qgis", "soma", "****")
49
50 choice = QMessageBox.question(None, "Vector Layer Analysis!",
51                               "Do you wish to do analysis by
                               putting Query on Database?",
                               QMessageBox.Yes | QMessageBox.No )
52
53
54 if choice == QMessageBox.Yes:
55     print("Yes, do Query on database!!")
56     # put the query box
57
58     (query, ok)= QDialog.getText(None, "Make a query",
59                                 "Query:", QLineEdit.Normal,
60                                 "ex. signal_strength =15")
61
62     if ok:
63         n = query
64         print("Your Query is: " + query)
65
66         # make a query database layer
67         uri.setDataSource("public", "route", "the_geom", query)
68         db_layer = QgsVectorLayer(uri.uri(), "database_layer", "
        postgres")
69         # adding to the map registry for display
70         QgsMapLayerRegistry.instance().addMapLayer(db_layer)
71     else:
72         print(" You cancelled Query! Try again!!")
73
74 else:
75
76     # making layer with no QUERY
77     uri.setDataSource("public", "route", "the_geom")
78     db_layer = QgsVectorLayer(uri.uri(), "database_layer", "postgres")
79     QgsMapLayerRegistry.instance().addMapLayer(db_layer)
80
81     """ adding layers to map canvas """
82
83 layers = QgsMapLayerRegistry.instance().mapLayers()
84 map_canvas_layer_list = [QgsMapCanvasLayer(i) for i in layers.values()]
85 db_canvas.setExtent(db_layer.extent())
86 db_canvas.setLayerSet(map_canvas_layer_list)
87
88 db_canvas.show()
89 db_qgs.exec_()

```

.3 Source Code: PyQGIS-Application

```

1 from PyQt4.QtWebKit import QWebView
2 from PyQt4 import QtGui, QtCore
3 from PyQt4.QtCore import *
4 from PyQt4.QtGui import *
5 from qgis.core import *
6 from qgis.utils import *
7 from qgis.gui import *
8 import numpy
9 import requests
10 import webbrowser
11 import sys
12 import os
13
14 class Gui_Window(QtGui.QMainWindow):
15     def __init__(self):
16         super(Gui_Window, self).__init__()
17         self.setGeometry(50,50,550,200)
18         self.setWindowTitle("Visualisation of Norge GIS Data")
19         self.setWindowIcon(QtGui.QIcon("./icons/ntnu-logo.png"))
20         background_img = QtGui.QImage("./icons/google1.jpg")
21         bg_palette = QtGui.QPalette()
22         role2 = QtGui.QPalette.Background
23         bg_palette.setBrush(role2, QtGui.QBrush(background_img))
24         self.setPalette(bg_palette)
25
26         # file menu
27         extractAction = QtGui.QAction("& Add CSV file !!", self)
28         extractAction.setShortcut("Ctrl+Q")
29         extractAction.setStatusTip("Leave the App")
30         extractAction.triggered.connect(self.add_csv_window)
31         icon = QtGui.QIcon()
32         icon.addPixmap(QtGui.QPixmap("./icons/ntnu-logo.png"))
33         self.statusBar()
34         mainMenu = self.menuBar()
35         fileMenu = mainMenu.addMenu("&File")
36         fileMenu.addAction(extractAction)
37         fileMenu.setIcon(icon)
38
39         # AboutMenu
40         extractAction_aboutMessageBox = QtGui.QAction("& What is NINU-
41 QGIS!!", self)
42         extractAction_aboutMessageBox.setStatusTip("About the App")
43         extractAction_aboutMessageBox.triggered.connect(self.
44 about_messageBox)
45         icon_qgis = QtGui.QIcon()
46         icon_qgis.addPixmap(QtGui.QPixmap("./icons/qgis.png"))
47         self.statusBar()
48         mainMenu = self.menuBar()
49         fileMenu = mainMenu.addMenu("&About")
50         fileMenu.addAction(extractAction_aboutMessageBox)
51         fileMenu.setIcon(icon_qgis)

```

```

50     self.home()
51
52     # new function
53
54     def home(self):
55
56         btn = QtGui.QPushButton("Quit", self)
57         btn.clicked.connect(self.close_application)
58         #btn.resize(100,100)
59         btn.resize(btn.minimumSizeHint())
60         btn.move(400,150)
61
62         # Toolbar
63         extractAction2 = QtGui.QAction(QtGui.QIcon("./icons/icon11.ico"
64 ), "Where am I?", self)
65         extractAction2.triggered.connect(self.whereami_toolbar)
66         self.toolBar = self.addToolBar("whereamiToolBar")
67         self.toolBar.addAction(extractAction2)
68
69         extractAction_webpage = QtGui.QAction(QtGui.QIcon("./icons/web.
70 png"), "Connect to NTNU-QGIS Webpage", self)
71         extractAction_webpage.triggered.connect(lambda: webbrowser.
72 open_new_tab("http://127.0.0.1/qgisweb/login"))
73         self.toolBar = self.addToolBar("
74 Toolbar_Upload_your_data_at_NTNU-QGIS_webpage")
75         self.toolBar.addAction(extractAction_webpage)
76
77         extractAction_DB = QtGui.QAction(QtGui.QIcon("./icons/postGIS.
78 png"), "Connect to PostgreSQL", self)
79         extractAction_DB.triggered.connect(lambda: webbrowser.
80 open_new_tab("http://localhost/phpgadmin/"))
81         self.toolBar = self.addToolBar("Toolbar_Connect_to_PostgreSQL")
82         self.toolBar.addAction(extractAction_DB)
83
84         extractAction_qgisServer = QtGui.QAction(QtGui.QIcon("./icons/
85 logo-qgis-server.png"), "Connect to QGIS Server", self)
86         extractAction_qgisServer.triggered.connect(lambda: webbrowser.
87 open_new_tab("http://qgis.ntnu.soma/cgi-bin/qgis_mapserv.fcgi"))
88         extractAction_qgisServer.triggered.connect(self.
89 close_application)
90         self.toolBar = self.addToolBar("Connect to QGIS Server")
91         self.toolBar.addAction(extractAction_qgisServer)
92
93         extractAction_wms = QtGui.QAction(QtGui.QIcon("./icons/globe.
94 png"), "Norge WMS Map", self)
95         extractAction_wms.triggered.connect(self.norge_wms_display)
96         self.toolBar = self.addToolBar("Extract WMS Norge")
97         self.toolBar.addAction(extractAction_wms)
98
99         extractAction_addLayer = QtGui.QAction(QtGui.QIcon("./icons/
100 layer-add.png"), "Create CSV vector layer", self)
101         extractAction_addLayer.triggered.connect(self.add_csv_window)

```

```

91     self.toolBar = self.addToolBar("Create vector layer")
92     self.toolBar.addAction(extractAction_addLayer)
93
94     extractAction3 = QtGui.QAction(QtGui.QIcon("./icons/layers-
icon3.png"), "Create PostGIS Vector layers", self)
95     extractAction3.triggered.connect(self.create_db_layer)
96     self.toolBar = self.addToolBar("Extract PostGIS Data")
97     self.toolBar.addAction(extractAction3)
98
99     self.progress = QtGui.QProgressBar(self)
100    self.progress.setGeometry(20, 230, 250, 20)
101
102    self.btn_progress = QtGui.QPushButton("Download Map", self)
103    self.btn_progress.resize(self.btn_progress.minimumSizeHint())
104    self.btn_progress.move(20,250)
105    self.btn_progress.clicked.connect(self.download)
106
107
108    print (self.style().objectName())
109
110    comboBox = QtGui.QComboBox(self)
111    comboBox.addItem("signal strength")
112    comboBox.addItem("Latitude")
113    comboBox.addItem("cde")
114    comboBox.move(400,300)
115    comboBox.activated[str].connect(self.style_choice)
116
117    # to display the GUI window on the screen , we use show()
118    self.show()
119
120
121    def close_application(self):
122        print("Hey! bye-bye!!")
123        sys.exit()
124
125    # About Dialog function
126    def about_messageBox(self):
127        QMessageBox.about(
128            None,
129            "About NTNU-QGIS",
130            "PostgreSQL/Postgis is used as a centralized geo-spatiotemporal
131            \
132            \
133            \
134            \
135            \
            data warehouse. Other databases and data sources are replicated
            at a regular interval into PostgreSQL, using various tools such
            as\
            OGR and FME. In the beginning NTNU-QGIS was mainly used as a
            viewer \
            of geodata on the geographical Maps and later more
            functionalities \
            added for doing deeper visual data analysis. \

```

```

136         The system is flexible and scalable to visualise data in real-
time \
137         connecting to ntnu-qgis database")
138
139     def norge_wms_display(self):
140         os.system("python norge_wms_display.py")
141
142     # add vector layer
143     def add_csv_window(self):
144         os.system("python add_csv_layer.py")
145
146     # creating layer from database
147     def create_db_layer(self):
148         os.system("python add_db_layer.py")
149
150     def style_choice(self, text):
151         QtGui.QApplication.setStyle(QtGui.QStyleFactory.create(text))
152
153     def enlarge_window(self, state):
154         if state == QtCore.Qt.Checked:
155             self.setGeometry(50,50,2000,1000)
156         else:
157             self.setGeometry(50,50,800,800)
158
159     def download(self):
160         self.completed = 0
161         while self.completed <100:
162             self.completed += 0.0001
163             self.progress.setValue(self.completed)
164
165
166     def close_messageBox(self):
167         choice = QtGui.QMessageBox.question(self, "Vector Layer
Analysis!",
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```
183 def main():
184     app = QtGui.QApplication(sys.argv)
185     GUI = Gui_Window()
186     app.exec_()
187
188
189 main()
```