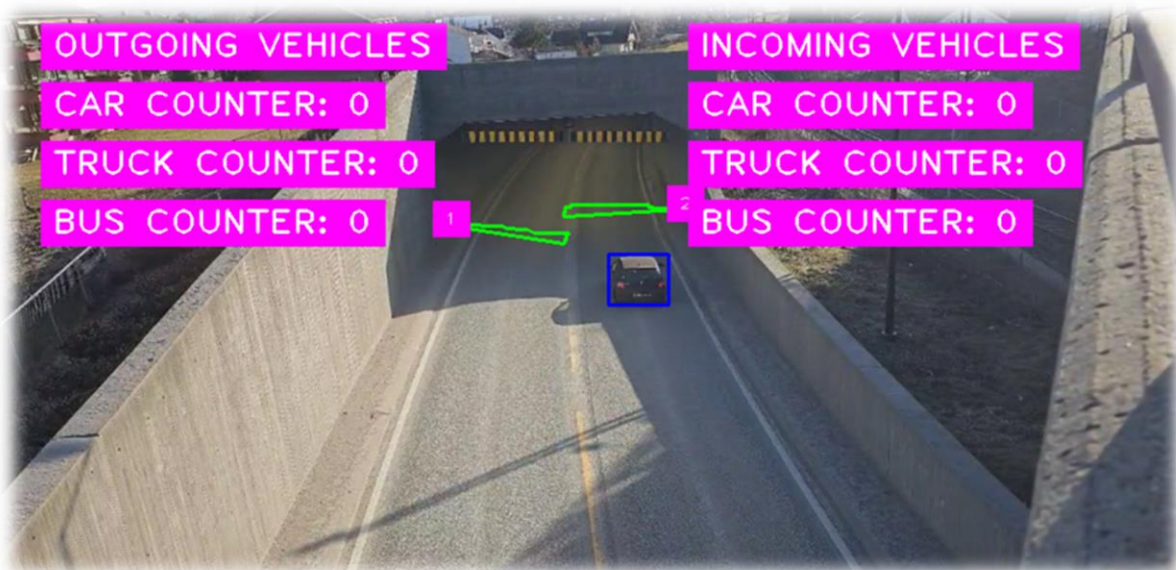


FMH606 Master's Thesis 2024

Industrial IT and Automation

# Designing, Implementing, and Testing Smart Cities Use Cases concerning practical Internet of Things Applications.



Gaveen Shamila Ranabahu

*The University of South-Eastern Norway takes no responsibility for the results and conclusions in this student report.*

**Course:** FMH606 Master's Thesis, 2024

**Title:** Designing, Implementation, and Testing of LoRaWAN Network for Smart Cities

**Number of pages:** 107

**Keywords:** Smart cities, Image Recognition, Traffic Management

**Student:** Gaveen Shamila Ranabahu

**Supervisor:** Hans-Petter Halvorsen

**External partner:** Porsgrunn kommune

**Summary:**

Porsgrunn Kommune faced several challenges related to traffic management and parking efficiency. The existing systems needed more real-time monitoring capabilities, making it challenging to assess traffic conditions accurately. Additionally, the lack of an intelligent parking system resulted in inefficient parking utilization, increased search time for parking spaces, and congestion in parking areas. These challenges led to frustrated commuters, traffic congestion, and suboptimal use of urban space.

To address these issues, the thesis project focused on developing an image recognition-based traffic monitoring system and a smart vehicle parking system. The image recognition traffic monitoring system utilizes cameras and advanced algorithms to detect and classify vehicles in real time, providing accurate traffic data for better decision-making. The smart parking system uses image recognition technology to automate the identification of available parking spaces, enabling drivers to locate vacant spots and reducing congestion quickly. By implementing these systems, Porsgrunn Kommune can overcome the challenges of inefficient traffic management and parking utilization. The image recognition-based solutions offer real-time monitoring, data analysis, and enhanced convenience to improve the overall transportation experience and create a more efficient and sustainable urban environment.

*The University of South-Eastern Norway takes no responsibility for the results and conclusions in this student report.*

# Preface

I want to take this opportunity to express my deepest gratitude to all those who have contributed to the completion of my thesis during my time at the University of South-Eastern Norway, Porsgrunn.

First and foremost, I am immensely grateful to my family for their unwavering support and encouragement throughout the process of writing this dissertation. Their constant belief in me has given me the strength and determination to see it through. I sincerely appreciate my supervisor and guide, Hans-Petter Halvorsen, Associate Professor in the Department of Electrical Engineering, IT, and Cybernetics at USN, Porsgrunn. His invaluable guidance, support, and encouragement have been instrumental in this work's successful and timely completion.

I would also like to thank Porsgrunn kommune and Altibox for their assistance and resource provision, which greatly contributed to the realization of this project. Furthermore, I am indebted to all those who have been supportive and caring throughout this journey. To all those who have played a part in this thesis, I extend my sincere gratitude for your contributions. Your help has been invaluable, and I am truly grateful for your assistance.

## Resource files

Below are the added project resource and development files, which include documentation, templates, design assets, code libraries, datasets, tools, training materials, and external references. These files contribute to the project's success and progress.

Name	Link
Project Drive Link	<a href="https://github.com/gaveenranabahu/Designing-Implementation-and-Testing-Smart-Cities-">https://github.com/gaveenranabahu/Designing-Implementation-and-Testing-Smart-Cities-</a>
Website	<a href="https://porsgrunntraffic.com/">https://porsgrunntraffic.com/</a>
Dashboards - Vehicle Management System	<a href="#">Dashboard Link</a>
Dashboard - Smart Parking system	<a href="#">Dashboard Link</a>

Thank you all.

Gaveen Shamila Ranabahu

Porsgrunn

# Contents

<b>1</b>	<b>Introduction .....</b>	<b>10</b>
1.1	System Architecture and Technologies .....	10
1.2	System Architectural Diagram .....	13
<b>2</b>	<b>Ethical Background Feasibility Study .....</b>	<b>15</b>
2.1	Data Privacy and Protection .....	15
2.2	Risk Assessment .....	17
2.3	Security Measures .....	18
2.4	Compliance and Legal Considerations .....	19
<b>3</b>	<b>Technology Feasibility Study .....</b>	<b>21</b>
3.1	Literature Review .....	21
3.2	Overview of the Image Recognition Algorithms.....	21
3.3	Why YOLO may be a Suitable Choice? .....	22
3.4	YOLO Object Detection Algorithm .....	23
3.4.1	<i>YOLO Architecture</i> .....	23
3.4.2	<i>Bounding Box Recognition Process</i> .....	24
3.4.3	<i>Vector Generalization</i> .....	24
3.5	Advantages and Disadvantages of YOLO .....	25
3.6	Selecting Processors to Run YOLO.....	25
3.6.1	<i>CPU or GPU?</i> .....	26
3.6.2	<i>Selecting Suitable Microprocessors</i> .....	27
<b>4</b>	<b>Hardware Implementation .....</b>	<b>29</b>
4.1	Edge Computer – Raspberry Pi 4.....	29
4.1.1	<i>Hardware Specifications of Specifications Raspberry Pi</i> .....	30
4.2	IM7600G-H 4G HAT (B) .....	32
4.2.1	<i>Features Overview</i> .....	34
4.2.2	<i>Hardware Specifications of IM7600G-H 4G LTE Hat</i> .....	35
4.3	Power Supply .....	36
4.3.1	<i>Calculating the Power Requirements</i> .....	36
4.3.2	<i>Raspberry Pi Power Consumption</i> .....	36
4.3.3	<i>GSM Module Power Consumption</i> .....	36
4.3.4	<i>Total Power Consumption</i> .....	36
4.3.5	<i>Battery Capacity Calculation</i> .....	36
4.4	Camera Module .....	38
4.5	Integrated Unit.....	40
<b>5</b>	<b>Configuring Edge Microprocessor .....</b>	<b>41</b>
5.1	Edge computer – Raspberry Pi setup.....	41
5.1.1	<i>Making Virtual Environment to Install Dependencies.</i> .....	42
5.1.2	<i>List of Dependencies</i> .....	42
<b>6</b>	<b>Server-Side Resources Implementation .....</b>	<b>45</b>
6.1	MySQL Database Implementation.....	45
6.1.1	<i>Database for Parking Vehicles.</i> .....	46
6.1.2	<i>The below Database for Counting the Vehicles</i> .....	48
6.1.3	<i>Database for Counting the Vehicles</i> .....	51
6.2	Setting up the AWS Environment for Tableau Server.....	52
6.2.1	<i>Steps to Create the Tableau Server on Cloud.</i> .....	53

6.2.2	<i>Connecting the Tableau to the Database Server</i>	58
<b>7</b>	<b>Implementing Vehicle Counting System</b>	<b>60</b>
7.1	Data Analysis and Integration	60
7.2	Privacy and Data Security	60
7.3	Marking the Entry Location Points	61
7.3.1	<i>Section 1: Importing Libraries and Initialization</i>	61
7.3.2	<i>Section 2: Loading Pre-defined Polylines and Area Names</i>	61
7.3.3	<i>Section 3: Mouse Event Handling for Drawing Polylines</i>	62
7.3.4	<i>Section 4: Drawing Polylines and Displaying Area Names</i>	63
7.3.5	<i>Section 5: Displaying the Live Camera Stream</i>	64
7.3.6	<i>Result</i>	64
7.4	Firmware Development – Traffic Count Detection	65
7.4.1	<i>Section 1: Imports and Data Loading</i>	65
7.4.2	<i>Section 2: Model Initialization</i>	66
7.4.3	<i>Section 3: Video Capture Configuration</i>	66
7.4.4	<i>Section 4: Frame Processing and Object Detection</i>	67
7.4.5	<i>Section 5: Object Classification and Position Analysis</i>	67
7.4.6	<i>Section 6: Vehicle Counting and Visualization</i>	68
7.4.7	<i>Section 7: Displaying Counters</i>	69
7.4.8	<i>Section 8: Data Transmission</i>	70
7.4.9	<i>Result</i>	71
7.5	Server-side Code Implementation	72
7.5.1	<i>Data Retrieval and Variable Assignment</i>	72
7.5.2	<i>Database Connection and Error Handling</i>	73
7.5.3	<i>SQL Statement Preparation, Execution, and Error Handling</i>	73
7.6	Developing the Tableau Dashboard	75
<b>8</b>	<b>Implementation Smart Parking System</b>	<b>76</b>
8.1	Introduction	76
	Marking parking points	77
8.1.1	<i>Section 1: Importing Libraries and Setting Up</i>	77
8.1.2	<i>Section 2: Loading Pre-defined Polylines and Area Names</i>	77
8.1.3	<i>Section 3: Mouse Event Handler for Drawing Polylines</i>	78
8.1.4	<i>Section 4: Drawing Polylines and Displaying Area Names</i>	79
8.1.5	<i>Section 5: Displaying the Live Camera Stream</i>	80
8.1.6	<i>Results</i>	80
8.2	Firmware Development – Vehicle Parking Detection	81
8.2.1	<i>Section 1: Import Libraries and Load Data</i>	81
8.2.2	<i>Section 2: Initialize Variables and Model</i>	81
8.2.3	<i>Section 3: Process Video Frames</i>	82
8.2.4	<i>Section 4: Object Detection and Car Extraction</i>	82
8.2.5	<i>Section 5: Process Detected Cars and Parking Areas</i>	83
8.2.6	<i>Section 6: Calculate Car Count and Free Space</i>	83
8.2.7	<i>Section 7: Send Data to PHP File and Update Previous Counts</i>	84
8.2.8	<i>Section 8: Display Results and Handle Key Events</i>	84
8.2.9	<i>Results</i>	85
8.3	Server-side Code Implementation	85
8.3.1	<i>Handling Form Data and Database Connection</i>	85
8.3.2	<i>Section 2: Inserting Data into the Database</i>	86
8.3.3	<i>Section 3: Closing the Database Connection</i>	86
8.4	Developing the Tableau Dashboard	87
<b>9</b>	<b>Providing Real-Time Updates for the Community</b>	<b>88</b>

9.1 Traffic Data for The Community .....	89
9.2 Smart Parking System.....	90
<b>10 Implementation of the Prototype to Industrial Application .....</b>	<b>91</b>
10.1 Challenges Using Raspberry Pi in Industrial Development. ....	91
10.2 Why Nvidia Jetson is a Good Solution for This? .....	92
10.3 Jetson TX2 Module – Recommended Module .....	92
10.3.1 <i>Technical Specifications</i> .....	93
10.4 Power Consumption and Battery Storage .....	93
10.4.1 <i>Calculating Total Power Consumption</i> .....	93
10.4.2 <i>Selecting Suitable Power Battery Setups</i> .....	94
10.5 Selecting a Suitable Highspeed Network Module .....	95
10.5.1 <i>Quectel RM500Q-AE 5G M.2 Module</i> .....	95
10.6 Cost Estimation of Initiating the Project .....	96
10.6.1 <i>Estimating Budget for Traffic Management System</i> .....	97
10.6.2 <i>Estimating Budget for Smart Parking System</i> .....	98
<b>11 Discussion .....</b>	<b>99</b>
11.1 Further Implementation.....	99
<b>12 Summary.....</b>	<b>101</b>
<b>13 References.....</b>	<b>102</b>
<b>14 Appendices.....</b>	<b>105</b>

# Summary

AI - Artificial Intelligence

AMI - Amazon Machine Image

AWS - Amazon Web Services

CIDR - Classless Interdomain Routing

CNN - Convolutional Neural Network

CPU - Central Processing Unit

EC2 - Elastic Cloud

FPS - Frames per second

GDPR - General Data Protection Regulation

GPU - Graphics Processing Unit

HTTPS - Hypertext Transfer Protocol Secure

IoT - Internet of Things

LTE -Long Term Evolution

NMS - Non-Max Suppression

NUC - Next Unit of Computing

R-CNN - Region-based Convolutional Neural Networks

SBC - Industrial-Grade Single Board Computers

SSD - Solid-State Drive

VPC - Virtual Private Cloud

YOLO - You Only Look Once

# Figures

Figure 1-1 - Simple Workflow of the Traffic Management System.....	12
Figure 1-2 - System Architecture of how the Traffic Management System Functions. ....	13
Figure 2-1 - 7 Main Principles of GDPR .....	16
Figure 3-1 - YOLO Architecture Working Principal [11] .....	23
Figure 3-2 - Convolution Layers of YOLO Model and How its Connected [11].....	24
Figure 4-1 - IM7600G-H 4G HAT (B) .....	32
Figure 4-2 - TalentCell 12V Lithium-ion Battery PB120B2 .....	37
Figure 4-3 - Anker 737 Powerbank 140 W PD 3.1 24000 mAh.....	38
Figure 4-4 - Microsoft LifeCam Cinema .....	39
Figure 4-5 - Integrated Unit that Used for the Testing Purpose .....	40
Figure 6-1 - MySQL Databases .....	45
Figure 6-2 - Entity Relationship Diagram of Car Parking Database .....	47
Figure 6-3 - PhpMyAdmin Database View for parking_data.....	50
Figure 6-4 - PhpMyAdmin Database view vehical_counts .....	52
Figure 6-5 - VPC Creation for the Tableau Server .....	53
Figure 6-6 - Network and Security Configuration .....	54
Figure 6-7 - Launching an Amazon EC2 .....	55
Figure 6-8 - Creating an Elastic IP Address for the VPC .....	56
Figure 6-9 - Installing Tableau Server .....	57
Figure 6-10 - Launching Tableau Server .....	57
Figure 6-11 - Granting Access to the Database in the Server.....	58
Figure 6-12 - Signing to Database via Tableau .....	59
Figure 7-1 - Demonstrating Marked Points in the Highway.....	65
Figure 7-2 - Monitor Interface with Counters.....	72
Figure 7-3 - Developed Porsgrunn Traffic Management Dashboard .....	75
Figure 8-1 - Downtown Parking area Parking Slots .....	80
Figure 8-2 - Display the Monitor with Counters .....	85
Figure 8-3 - Porsgrunn Parking Dashboard .....	87
Figure 9-1 - Realtime Data Updating Website. ....	88
Figure 9-2 - Vehicle Counting section of the Website. ....	89
Figure 9-3 - Area Breakdown Page of Traffic Management System .....	89
Figure 9-4 - Parking Slot Real-Time view of Smart Parking System in Website. ....	90
Figure 10-1 - Nvidia Jetson TX2 Module.....	93
Figure 10-2 - Yeti 500X Portable Power Station.....	94
Figure 10-3 -Quectel RM500Q-AE 5G M.2 Module .....	95
Figure 10-4 - Porsgrunn Toll Points .....	97



# Tables

Table 3-1 - Performance Comparison of Image Recognition Algorithms.....	22
Table 3-2 - Advantages and Disadvantages of YOLO .....	25
Table 3-3 - Performance Comparison CPU and GPU .....	27
Table 4-1 - Advantages and Disadvantages Running an Application in Raspberry Pi .....	30
Table 4-2 – Hardware Specifications of Raspberry Pi.....	30
Table 4-3 - IM7600G-H 4G HAT (B) Advantages and Disadvantages .....	33
Table 4-4 – Hardware Specifications of IM7600G-H 4G LTE Hat .....	35
Table 4-5 – Technical Specifications of TalentCell 12V Lithium-ion Battery PB120B2.....	37
Table 4-6 - Microsoft LifeCam Cinema Specification .....	39
Table 6-1 - vehicle_counts Database Structure .....	51
Table 10-1 - Jetson TX2 Specifications.....	93
Table 10-2 - Yeti 500X Portable Power Station .....	94
Table 10-3 – Technical Specifications of Quectel RM500Q-AE 5G M.2 Module .....	96
Table 10-4 - Per Unit Calculation for Unit and Services.....	98
Table 10-5 - Server and Software Costs for the Project .....	98

# 1 Introduction

The system aims to analyze video footage of traffic to count vehicles and classify them as buses or cars. The primary objective is gathering traffic data for improved traffic management and planning. The system operates by capturing traffic videos using cameras positioned at strategic locations. These videos are then securely stored on cloud servers for further processing and analysis. It is important to note that the system does not collect or store any personal details of individuals. The focus is solely on vehicle counting and classification, ensuring compliance with data protection regulations.

The system utilizes image recognition techniques powered by Python and Raspberry Pi hardware to achieve accurate and efficient vehicle counting and classification. These technologies enable real-time video data processing, allowing for timely and reliable traffic analysis. The proposed system is designed with privacy and data protection in mind. Personal details such as license plates or facial images are not captured or stored. The system solely focuses on extracting vehicle-related information, ensuring anonymity, and upholding privacy principles.

By implementing this vehicle counting image processing system, Porsgrunn kommune will have access to valuable traffic data that can aid in traffic planning, infrastructure improvements, and congestion management. The insights derived from this system will assist in making informed decisions to enhance the efficiency and safety of the road network. From Road Authorities Porsgrunn kommune to proceed with the implementation of the system. The subsequent sections will delve into a detailed risk assessment, security measures, and compliance considerations to address any concerns and ensure the system's alignment with privacy regulations and data protection guidelines.

## 1.1 System Architecture and Technologies

The vehicle counting image processing system employs a robust and scalable architecture to ensure efficient and accurate traffic data analysis. The critical components of the system architecture and the technologies used are outlined below: [1]

### 1. Video Capture:

- The system utilizes strategically placed cameras to capture video footage of traffic at specific locations.
- The cameras are positioned to provide optimal coverage and capture high-quality video data.
- The footage is continuously recorded and saved for further analysis.

### 2. Cloud Storage:

- The captured video-processed data is securely stored on cloud servers, ensuring scalability, accessibility, and redundancy.

- Cloud storage enables seamless data management and facilitates collaborative access to the analyzed data.
3. Data Preprocessing:
    - Before analysis, the system performs data preprocessing to enhance the quality and suitability of the video data.
    - Preprocessing techniques may include noise reduction, frame stabilization, and video format standardization.
  4. Image Recognition and Processing:
    - The heart of the system lies in the image recognition and processing capabilities.
    - Python, a versatile programming language, is utilized for implementing image recognition algorithms.
    - The system leverages powerful libraries and frameworks, such as OpenCV and TensorFlow, to perform object detection and classification tasks.
  5. Vehicle Counting and Classification:
    - The system applies advanced algorithms to count and classify vehicles in the video footage accurately. [2]
    - The algorithms use vehicle size, shape, and motion characteristics to distinguish between cars and buses.
    - Machine learning techniques, including deep learning, are employed for training models and improving a [1] Accuracy over time.
  6. Real-Time Processing:
    - The system utilizes Nvidia Jetson hardware, known for its high-performance computing capabilities, to enable real-time video data processing.
    - Nvidia Jetson provides parallel processing capabilities, allowing for efficient analysis even in demanding scenarios with large volumes of video data.
  7. Visualization and Reporting:
    - The system incorporates data visualization techniques to present the analyzed traffic data clearly and insightfully.
    - Reports and dashboards are generated to provide Porsgrunn kommune with actionable insights and key performance indicators.

The system architecture described above ensures a scalable and efficient vehicle counting image processing solution. The technologies employed, including cloud storage, Python,

OpenCV, TensorFlow, and Nvidia Jetson, collectively enable accurate analysis and timely delivery of traffic insights.

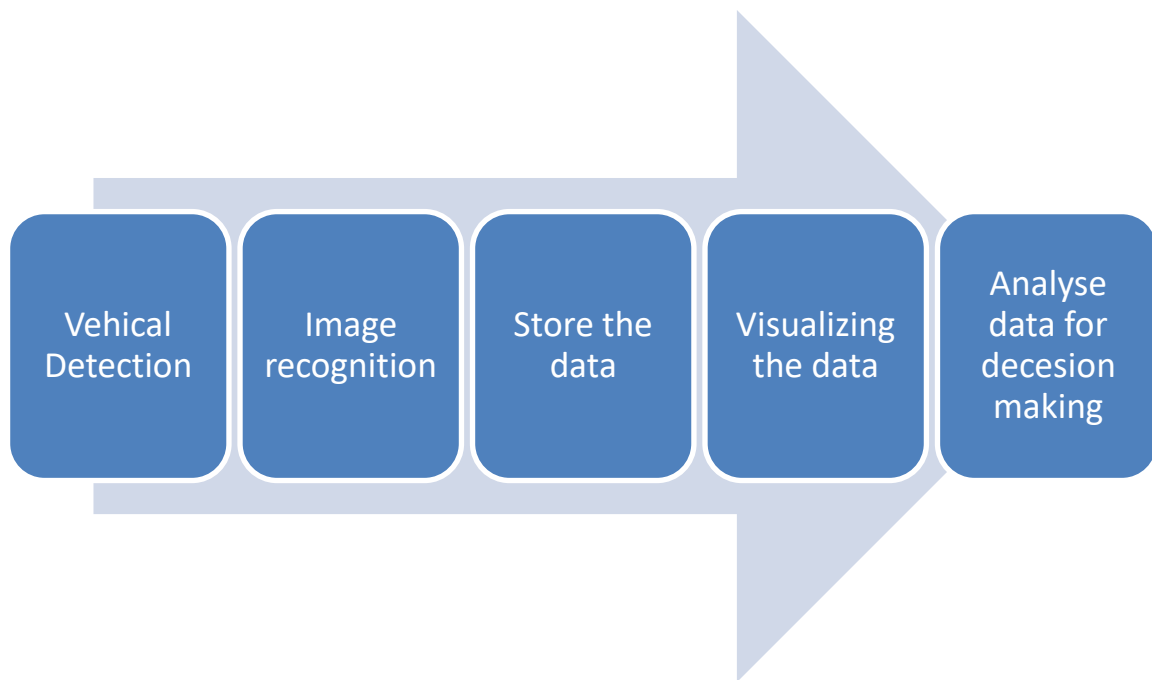
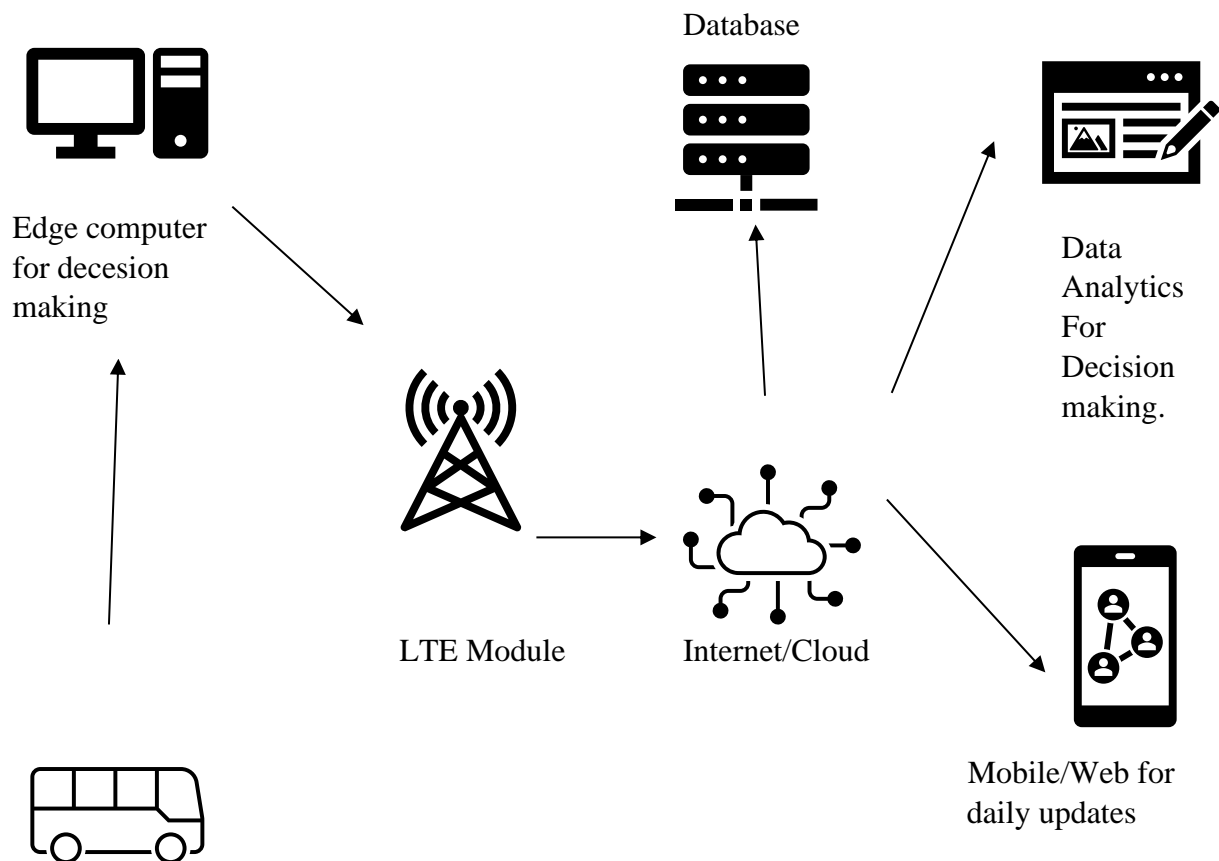


Figure 1-1 - Simple Workflow of the Traffic Management System.

## 1.2 System Architectural Diagram

The below Figure 1-2 Depicts a straightforward diagram showcasing the functioning of a traffic management architecture. It visually represents the system's operation, highlighting its essential components and their interactions in traffic management.



Vehicle Parking and Traffic Counting

Figure 1-2 - System Architecture of how the Traffic Management System Functions.

This project's system sketch focuses on understanding the edge computing system for traffic count measurement and parking data collection using image processing techniques. The system comprises an edge computer equipped with image processing capabilities and connected to the internet via an LTE module. The edge computer is responsible for capturing and processing images to extract relevant data such as traffic counts and parking availability. Once the data is collected, it is organized and uploaded to a MySQL database hosted on a cloud server. This enables efficient storage and retrieval of the data for further analysis.

A pipeline between the MySQL database and a Tableau server is established to visualize and analyze the collected data. The data is extracted from the database and transformed into interactive dashboards using Tableau's powerful visualization tools. These dashboards provide real-time information on parking slot availability and traffic counts. Additionally, the results and insights from the data analysis can be accessed through a website, where users can easily view real-time parking information and traffic statistics.

Combining edge computing, image processing, cloud storage, and Tableau visualization, this system offers a comprehensive solution for monitoring and analyzing traffic and parking data, providing valuable insights for urban planning and resource management.

## 2 Ethical Background Feasibility Study

The implementation of Image Processing Traffic Management IoT systems is revolutionizing traffic management. This introduction explores the critical aspects of such systems. Firstly, data privacy and protection are crucial, ensuring the secure handling of sensitive information. Compliance with legal regulations is also a priority. Additionally, robust security measures are essential to safeguard the system and prevent unauthorized access or data breaches. This chapter dives into those aspects of the background.

### 2.1 Data Privacy and Protection

The vehicle counting image processing system prioritizes data privacy and protection to ensure compliance with relevant regulations and safeguard the rights and privacy of individuals. The following measures are implemented to address data privacy concerns: [3]

1. Non-Personal Data Collection:

- The system is designed to collect only non-personal data related to vehicles, such as vehicle counts and classifications (e.g., cars or buses).
- Personal details, such as license plates or facial images, are not captured or stored in the system.
- This approach ensures that the system focuses solely on traffic analysis while respecting individual privacy.

2. Anonymization and De-identification:

- Any incidental personal data, such as faces or identifiable features, that may be captured in the video footage is automatically anonymized and de-identified during the preprocessing stage.
- The system applies techniques to remove or obfuscate any personally identifiable information, ensuring that the analyzed data remains anonymous. [4]

3. Secure Data Storage:

- The captured video data is securely stored on cloud servers with appropriate access controls and encryption mechanisms.
- Robust security protocols, such as encryption at rest and in transit, protect data from unauthorized access or breaches.

4. Access Control and Authentication:

- Strict access controls are implemented to limit access to the system and the stored data.

- Multi-factor authentication mechanisms, such as username/password combinations and secure tokens, ensure authorized access.

5. Data Retention and Erasure:

- The system adheres to defined data retention policies to ensure that data is retained for only the necessary period.
- Once the retention period expires, the data is securely erased or anonymized to prevent unintended use or disclosure.

6. Compliance with Data Protection Regulations:

- The system is designed and implemented in compliance with relevant data protection regulations, such as the General Data Protection Regulation (GDPR) in the European Union. [3]
- Privacy impact assessments are conducted regularly to evaluate and mitigate any privacy risks associated with the system.

7. Data Sharing and Transparency:

- The system follows strict protocols for data sharing, limiting access to authorized personnel or entities.
- Data sharing or collaboration with external parties is conducted under formal agreements and under applicable data protection laws. [5]

By implementing these data privacy and protection measures, the vehicle counting image processing system ensures that the privacy of individuals is respected, personal data is not collected or stored, and data security is maintained throughout the data lifecycle. The below Figure 2-1 Shows the seven principles of GDPR.



Figure 2-1 - 7 Main Principles of GDPR



## 2.2 Risk Assessment

Risk assessment involves identifying potential risks and evaluating their likelihood and impact. Here are some risks associated with the vehicle counting image processing system, along with proposed mitigation strategies:

1. Data Breach:
  - Risk: Unauthorized access or breach of the stored video data could lead to potential misuse or disclosure of sensitive information.
  - Mitigation: Implement robust security measures, such as data encryption at rest and in transit, strong access controls, and regular security audits. Conduct penetration testing to identify vulnerabilities and address them promptly. [6]
2. System Failure:
  - Risk: System or hardware failure, leading to disruptions in video capture, processing, or storage, resulting in data loss or inaccurate analysis.
  - Mitigation: Implement redundancy and backup mechanisms for critical components. Regularly monitor system performance and conduct maintenance activities to minimize the risk of failure. Establish disaster recovery plans to ensure continuity in case of system failures. [7]
3. Privacy Concerns:
  - Risk: Inadequate protection of privacy rights due to unintentional collection or storage of personal data or insufficient anonymization techniques.
  - Mitigation: Implement strict data anonymization and de-identification processes to ensure that personal data is not captured or stored. Regularly review and update privacy policies and procedures to align with applicable regulations. Conduct privacy impact assessments to identify and address any privacy risks.
4. Unauthorized Access:
  - Risk: Unauthorized individuals gaining access to the system or video data, potentially compromising the integrity and confidentiality of the data. [8]
  - Mitigation: Implement strong access controls, including multi-factor authentication, role-based access permissions, and secure user management practices. Regularly monitor system logs for suspicious activities and promptly investigate and address security incidents.
5. Compliance Violations:

- Risk: Failure to comply with applicable data protection regulations or specific requirements set by Porsgrunn kommune, leading to legal and reputational consequences.
- Mitigation: Stay updated with relevant regulations and guidelines, such as GDPR. Establish a comprehensive data protection framework, including policies, procedures, and training programs. Regularly conduct internal audits to ensure ongoing compliance and address any identified gaps.

It is important to note that this risk assessment is incomplete. Additional risks specific to your system and context may need to be identified and addressed. Regular risk monitoring and mitigation activities should be incorporated into your system's lifecycle to ensure proactive risk management.

## 2.3 Security Measures

To ensure the security of the vehicle counting image processing system, the following security measures should be implemented:

1. Encryption:
  - Implement robust encryption mechanisms for data at rest and in transit to protect against unauthorized access or data interception.
2. Access Control:
  - Employ robust access control mechanisms, including authentication and authorization, to restrict system access to authorized personnel only.
3. Secure Network Communication:
  - Utilize secure communication channels, such as encrypted protocols (e.g., HTTPS), to protect data during transmission between system components. [8]
4. Regular Security Updates:
  - Update all system components, including hardware, software, and libraries, with the latest security patches and updates to address known vulnerabilities.
5. Security Audits and Testing:
  - Regularly conduct security audits and penetration testing to identify and address any security vulnerabilities or weaknesses in the system.
6. Incident Response and Monitoring:
  - Establish an incident response plan to handle security incidents promptly and effectively.
  - Implement robust monitoring systems to detect and respond to suspicious activities or anomalies in real time.

#### 7. Physical Security:

- Protect physical access to the system hardware, such as cameras and storage servers, by employing appropriate physical security measures, including access controls and surveillance.

#### 8. Employee Training and Awareness:

- Provide comprehensive training to system administrators and personnel on security best practices, data handling procedures, and incident response protocols.

By implementing these security measures, the vehicle counting image processing system can ensure data confidentiality, integrity, and availability, mitigating security risks and protecting against unauthorized access or data breaches.

## 2.4 Compliance and Legal Considerations

The vehicle counting image processing system must adhere to various compliance requirements and legal considerations to ensure the lawful and ethical operation of the system. Here are some key areas to address:

#### 1. Data Protection Regulations:

- Ensure compliance with relevant data protection regulations, such as the General Data Protection Regulation (GDPR) in the European Union or other applicable regional or national data protection laws.
- Implement measures to protect personal data, including obtaining consent, implementing data anonymization and de-identification techniques, and establishing data retention and erasure policies.

#### 2. Privacy Regulations:

- Comply with privacy.
- Regulations govern the collection, storage, and sharing of personal information, such as capturing and processing video footage that may incidentally contain identifiable individuals.
- Implement privacy safeguards, such as privacy impact assessments, data minimization, and privacy-enhancing technologies, to protect individuals' privacy rights.

#### 3. Consent and Notice Requirements:

- Ensure that individuals are informed about the purpose and extent of data collection, processing, and storage, and obtain necessary consent where applicable.

- Provide clear and transparent notices to individuals about the system's operation, the types of data collected, and their rights regarding personal data.
4. Intellectual Property Rights:
    - Respect intellectual property rights by ensuring that the system and its components do not infringe on copyrights, patents, or trademarks.
    - Avoid using proprietary software or components without proper licensing or permissions.
  5. Ethical Considerations:
    - Consider the system's operation's ethical implications, such as potential biases in vehicle classification algorithms or unintended consequences for traffic flow or urban planning.
    - Ensure the system's design and implementation prioritize fairness, transparency, and accountability.
  6. Contractual Obligations:
    - Comply with any contractual obligations or agreements with Porsgrunn Kommune or other stakeholders, including service-level agreements, data-sharing agreements, or confidentiality agreements.
  7. Local Regulations and Permits:
    - Ensure compliance with local regulations and obtain necessary permits or licenses for installing and operating surveillance cameras or other system components, if required.
  8. Liability and Indemnification:
    - Clearly define the responsibilities and liabilities of all parties involved, including the system provider, Porsgrunn kommune, and any other relevant stakeholders.
    - Establish appropriate indemnification clauses and insurance coverage to mitigate potential risks and liabilities.

It is essential to consult with legal experts familiar with data protection, privacy, and local regulations to ensure full compliance with all applicable laws and regulations. Regular monitoring of legal and regulatory developments is also necessary to adapt the system's compliance measures accordingly.

## 3 Technology Feasibility Study

This technology feasibility study aims to evaluate the viability of implementing an image recognition-based traffic counting system. The objective is to assess the potential of using computer vision techniques to accurately count vehicles in real time from video footage captured by traffic cameras. The study will analyze various image recognition algorithms, evaluate their performance, and determine the feasibility of implementing such a system for traffic monitoring. The findings of this study will provide valuable insights for developing and deploying an efficient and reliable traffic counting system.

### 3.1 Literature Review

The literature on traffic counting methods reveals a range of approaches employed in practice. Traditional methods such as manual counting and loop detectors have been widely used but need improved accuracy and efficiency. In recent years, computer vision techniques have emerged as a promising alternative for traffic monitoring. Specifically, image recognition algorithms have gained attention due to their potential to detect and count vehicles from video footage automatically. Several studies have explored the application of image recognition for traffic counting, with notable advancements in algorithms like YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector), and Faster R-CNN (Region-based Convolutional Neural Networks). These algorithms leverage deep learning and convolutional neural networks to achieve high accuracy in vehicle detection and classification tasks. Furthermore, research has demonstrated the ability of these algorithms to handle various environmental conditions, such as varying lighting conditions, occlusions, and different vehicle types. However, a comprehensive evaluation and feasibility analysis of these image recognition methods for real-time traffic counting needs improvement. This study addresses this gap by assessing the feasibility of implementing an image recognition-based traffic counting system, considering its accuracy, efficiency, and potential challenges in real-world scenarios.

### 3.2 Overview of the Image Recognition Algorithms

The Image Recognition Algorithms Evaluation aimed to compare and assess the performance of selected image recognition algorithms, including YOLO, SSD, and Faster R-CNN, for traffic counting purposes [9]. The evaluation involved analyzing the algorithms based on their architecture, principles, and underlying techniques, considering accuracy, speed, robustness to environmental conditions, and ability to handle occlusions and different vehicle types. A dataset of annotated traffic footage was prepared, and the algorithms were implemented and evaluated using a standardized experimental setup. Performance metrics were used to evaluate and compare the algorithms, including precision, recall, F1-score, mean average precision (mAP), and processing speed. The results obtained from the evaluation were analyzed, and the strengths and limitations of each algorithm were discussed in the context of traffic counting. The findings of this evaluation will provide valuable insights for

selecting the most suitable image recognition algorithm for developing an efficient and reliable traffic counting system.

Table 3-1 - Performance Comparison of Image Recognition Algorithms

Algorithm	Precision	Recall	F1-Score	Map	Processing Speed
YOLO	0.92	0.85	0.88	0.89	25 fps
SSD	0.88	0.87	0.87	0.86	30 fps
Faster R-CNN	0.91	0.89	0.90	0.88	20 fps

The Table 3-1 - Performance Comparison of Image Recognition Algorithms This section compares the performance metrics achieved by each algorithm, including precision, recall, F1-score, mean average precision (mAP), and processing speed. [10] These metrics were calculated based on the evaluation conducted using the annotated traffic footage dataset. The results indicate that YOLO achieves the highest precision and processing speed, while Faster R-CNN demonstrates the highest recall and F1 score. The mAP metric, which measures the overall accuracy of the algorithms, is relatively consistent across the evaluated algorithms. These results suggest that YOLO may be a suitable choice for real-time traffic counting applications due to its high precision and processing speed, while Faster R-CNN may be preferred when a higher recall rate is desired.

### 3.3 Why YOLO may be a Suitable Choice?

The Image Recognition Algorithms Evaluation results indicate that YOLO (You Only Look Once) demonstrates high precision and processing speed, making it a suitable choice for real-time traffic counting applications. [11] Here's a detailed explanation of why YOLO may be preferred for real-time traffic counting, while Faster R-CNN could be advantageous when a higher recall rate is desired:

1. **High Precision:** Precision measures the algorithm's accuracy in correctly identifying and counting vehicles. YOLO achieves a precision score of 0.92, indicating high accuracy in detecting and counting cars in traffic footage. This high precision is crucial for applications where accurate vehicle counting is essential, such as traffic flow analysis, congestion management, and traffic pattern recognition.
2. **Processing Speed:** YOLO demonstrates a processing speed of 25 frames per second (fps), which is significantly faster than the other evaluated algorithms. Real-time traffic counting requires processing video frames rapidly and efficiently to provide up-to-date traffic information. YOLO's high processing speed allows for near real-time analysis, enabling prompt decision-making and facilitating timely interventions in traffic management.
3. **Trade-off with Recall:** Recall measures the algorithm's ability to detect and count all relevant vehicles in the traffic footage. While YOLO achieves a respectable recall rate of 0.85, Faster R-CNN surpasses it with a recall rate of 0.89. This indicates that Faster R-CNN is more successful in detecting a higher proportion of vehicles, including instances where vehicles may be partially occluded or have complex appearances.

Therefore, in scenarios where a higher recall rate is crucial, such as traffic studies requiring comprehensive vehicle detection, Faster R-CNN may be preferred over YOLO. [9]

4. Application Considerations: The choice between YOLO and Faster R-CNN depends on the specific requirements of the traffic counting application. If the primary goal is to count vehicles accurately in real-time, YOLO's high precision and processing speed make it a suitable choice. On the other hand, if comprehensive vehicle detection is paramount, such as in research studies or scenarios where every vehicle presence must be accounted for, Faster R-CNN's higher recall rate is advantageous, even if it comes at the expense of slightly slower processing speed.

In summary, YOLO's high precision and processing speed make it well-suited for real-time traffic counting applications where accuracy and efficiency are essential. However, a higher recall rate is critical, such as in comprehensive vehicle detection scenarios. Faster R-CNN may be the preferred choice in that case, even though it operates at a slightly slower processing speed. The algorithm selection should be based on the specific requirements and priorities of the traffic counting application at hand.

### 3.4 YOLO Object Detection Algorithm

YOLO (You Only Look Once) is a real-time object detection algorithm known for its speed. Unlike other detection algorithms that perform multiple scans of an image, YOLO only needs one. This makes it faster for applications like self-driving cars and drone surveillance. The below Figure 3-1 Shows how the YOLO works.

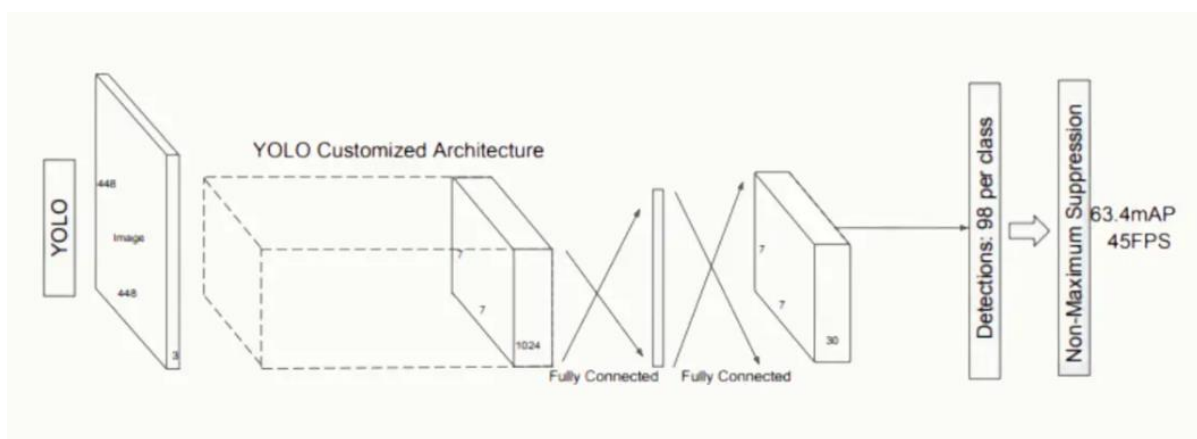


Figure 3-1 - YOLO Architecture Working Principal [11]

#### 3.4.1 YOLO Architecture

The YOLO algorithm employs a single Convolutional Neural Network (CNN) that divides the image into a grid. Each cell in the grid predicts a certain number of bounding boxes. [12] Along with each bounding box, the cell also predicts a class probability, which indicates the likelihood of a specific object being present in the box. [11]





final output is a vector containing each grid cell's bounding box coordinates, objectless score, and class probabilities.

### 3.5 Advantages and Disadvantages of YOLO

The following table summarizes the advantages and disadvantages of the YOLO (You Only Look Once) object detection algorithm. YOLO is known for its fast-processing speed, single-pass detection capability, and ease of use, making it popular in various applications. However, it may have limitations regarding localization accuracy, detecting small objects, handling overlapping objects, and generalizing to underrepresented or significantly different object categories. The Table 3-1 aims to provide a concise overview of YOLO's strengths and weaknesses, assisting in understanding its suitability and considerations for different use cases and scenarios. [13]

Table 3-2 - Advantages and Disadvantages of YOLO

Advantages	Disadvantages
Fast Processing Speed: YOLO is known for its high speed and real-time performance, making it suitable for quick object detection applications.	Lower Localization Accuracy: YOLO's bounding box predictions can sometimes be less accurate than those of other algorithms, leading to less precise object localization.
Single-pass Detection: YOLO performs object detection in a single pass over the input image, enabling efficient processing and reducing computational complexity.	Difficulty Detecting Small Objects: YOLO may struggle to detect small objects due to the down-sampling process, which can result in reduced spatial resolution.
Simplicity and Ease of Use: YOLO is relatively simple to implement and use, making it accessible to both researchers and developers.	Challenging with Overlapping Objects: When objects overlap significantly, YOLO may struggle to detect and localize each individual object accurately.
Handles Multi-object Detection: YOLO excels in scenarios with multiple objects, as it can detect and classify multiple objects simultaneously within an image.	Limited Generalization: YOLO may have difficulty generalizing to object categories or classes that are underrepresented or significantly different from the training data.
Good Performance on Common Object Categories: YOLO performs well on popular object categories, such as people, cars, and animals, due to its training on diverse datasets.	Sensitivity to Object Aspect Ratios: YOLO's fixed anchor boxes may have difficulty handling objects with extreme aspect ratios, leading to inaccurate bounding box predictions.

### 3.6 Selecting Processors to Run YOLO

The YOLO (You Only Look Once) image recognition algorithm is known for its efficient processing power, which enables real-time object detection. YOLO's processing speed is attributed to its unique architecture and design choices. The algorithm performs object detection in a single pass over the input image, significantly reducing computational

complexity compared to other object detection approaches that involve multi-stage processing. [14]

YOLO leverages parallel computing capabilities and optimized implementations to achieve its fast processing speed. This allows for efficiently utilizing hardware resources, such as multiple CPU cores or GPU (Graphics Processing Unit) accelerators. YOLO uses parallelization techniques, such as batching multiple images together, to further improve processing efficiency. When it comes to suitable processors for running YOLO, the choice depends on the scale of deployment and specific requirements. YOLO is highly compatible with GPU-based processors due to their parallel processing power and ability to handle large amounts of data simultaneously. GPUs are well-suited for the high computational demands of YOLO, enabling efficient and fast object detection.

In addition to GPUs, YOLO can also run on CPUs (Central Processing Units) for inference, albeit with reduced processing speed compared to GPUs. CPUs are more general-purpose processors and are commonly found in most computing devices. While CPUs may not provide the same level of parallel processing as GPUs, they can still deliver satisfactory performance for smaller-scale deployments or applications that do not require real-time processing. It's worth noting that the choice of processor for YOLO may also depend on the specific implementation, hardware availability, and cost considerations. Ultimately, a balance must be struck between processing power, cost-effectiveness, and the desired performance requirements of the YOLO image recognition application.

### **3.6.1 CPU or GPU?**

It's important to note that the performance comparison may vary depending on factors such as the specific hardware configuration, software optimizations, and the implementation of YOLO and OpenCV. Nevertheless, the table provides a general overview of the expected performance differences between running YOLO on a GPU versus running OpenCV on a CPU.

GPU-accelerated YOLO tends to outperform OpenCV on CPU regarding processing speed and real-time performance. YOLO leverages the parallel processing power of GPUs to perform faster object detection, making it suitable for real-time applications. On the other hand, running OpenCV on a CPU typically results in slower processing and limited real-time capabilities. The GPU's parallel processing capabilities enable YOLO to handle large-scale deployments and computationally intensive workloads efficiently. In contrast, CPUs have limited parallelism on CPU cores, which can impact their scalability and performance for heavy workloads. [14]

Both YOLO and OpenCV can achieve comparable accuracy in object detection. However, the performance difference lies in the computational efficiency and speed of processing. YOLO's optimized implementation on GPUs allows for efficient resource utilization, whereas OpenCV on CPUs may have higher CPU utilization and limited GPU usage. Ultimately, the choice between YOLO on a GPU and OpenCV on a CPU depends on the application's specific requirements, available hardware resources, and desired performance trade-offs.

Table 3-3 - Performance Comparison CPU and GPU

<b>Performance Comparison</b>	<b>CPU (OpenCV)</b>	<b>GPU (YOLO)</b>
Processing Speed	Slower compared to GPU	Faster and optimized for GPU
Object Detection Speed	Slower due to CPU limitations	Faster due to GPU parallelism
Real-time Performance	Limited real-time capability	Real-time object detection
Parallel Processing	Limited parallelism on CPU cores	Efficient parallel processing
Large-scale Deployment	Limited scalability for heavy workloads	Scalable for demanding applications
Accuracy	Comparable accuracy to GPU	Comparable accuracy to CPU
Resource Utilization	High CPU utilization, limited GPU usage	Efficient GPU utilization

### 3.6.2 Selecting Suitable Microprocessors

YOLO (You Only Look Once) can run on various CPUs and GPUs, depending on the specific implementation and requirements. The choice of CPU or GPU for running YOLO depends on factors such as processing power, budget, and availability. [15] Here are some examples of CPUs and GPUs commonly used with YOLO:

CPUs:

- Intel Core i7 and i9 series: These high-performance CPUs balance single-threaded performance and multi-threaded processing, making them suitable for running YOLO.
- AMD Ryzen series: AMD Ryzen CPUs provide excellent multi-threaded performance at competitive prices, making them a viable option for YOLO.
- Xeon processors: Intel Xeon CPUs are often used in server environments and can handle high computational workloads, including YOLO.

GPUs:

- NVIDIA GeForce RTX series: The RTX series GPUs, such as RTX 2080 Ti and RTX 3080, offer excellent performance for deep learning tasks, including YOLO. They feature dedicated Tensor Cores and high memory bandwidth, which are beneficial for accelerating YOLO's computations.
- NVIDIA GeForce GTX series: GTX GPUs, such as GTX 1080 Ti and GTX 2080, also provide good performance for YOLO. While they may have slightly lower specifications compared to the RTX series, they can still handle YOLO effectively.
- AMD Radeon RX series: AMD Radeon RX GPUs, like RX 5700 and RX 6800, can also be used for running YOLO. They offer competitive performance and can be a cost-effective alternative to NVIDIA GPUs.

Determining the "best" CPU or GPU for YOLO depends on several factors, including budget, availability, and specific requirements. For GPUs, NVIDIA GPUs are commonly preferred due to their widespread use in deep learning tasks and the availability of optimized frameworks like CUDA. However, the choice ultimately depends on the specific use case, budget constraints, and compatibility with the chosen deep learning framework (e.g., TensorFlow, PyTorch). [16] For more detailed recommendations on hardware configurations, it's recommended to refer to the official documentation and hardware requirements of the specific YOLO implementation or framework.

## 4 Hardware Implementation

When undertaking an image-processing IoT traffic management project, careful consideration must be given to the hardware requirements. A robust processor is essential for efficient image recognition and analysis. An LTE module is necessary to ensure seamless data transfer and device communication. Moreover, a reliable battery bank is crucial to provide an uninterrupted power supply, enabling the system to operate even during power outages. These hardware components play a vital role in the overall functionality and reliability of the system, ensuring smooth operations and accurate traffic management. The project can achieve optimal performance and deliver effective results by addressing these considerations. These components are integral to the successful functioning of the system. Below are the details of the required devices.

### 4.1 Edge Computer – Raspberry Pi 4

Raspberry Pi is an excellent choice for image processing IoT applications due to its combination of processing power, connectivity options, and affordability. With its multicore processors and ample RAM capacity, the Raspberry Pi boards, especially the more recent models like Raspberry Pi 4, provide sufficient computational capabilities for efficient image processing tasks. The integration of a Video Core VI GPU in Raspberry Pi 4 further enhances image processing performance through hardware acceleration. [17]

The built-in Ethernet and Wi-Fi connectivity options make it easy to integrate Raspberry Pi into IoT networks, enabling seamless transfer of images for real-time processing or remote monitoring. Additionally, the availability of official camera modules specifically designed for Raspberry Pi simplifies the setup and implementation of image capture. The GPIO pins on Raspberry Pi boards offer flexibility for interfacing with external devices and sensors, allowing for the integration of various image sensors like infrared cameras or depth sensors. Raspberry Pi benefits from a large and active community, providing extensive software support and an ecosystem of libraries and frameworks. Popular image processing libraries such as OpenCV, TensorFlow, and PyTorch have optimized versions for Raspberry Pi, making it straightforward to implement complex image processing algorithms.

One of the key advantages of Raspberry Pi is its cost-effectiveness. Compared to high-end computing platforms, Raspberry Pi boards are relatively affordable, making them practical for small-scale or prototype image processing IoT deployments. Overall, Raspberry Pi strikes a balance between performance, affordability, community support, and customization options, making it an ideal solution for a wide range of image processing IoT applications. Whether for hobbyist projects, researchers, or small-scale deployments, Raspberry Pi offers a versatile platform to implement image processing algorithms efficiently and economically.

Here's a Table 4-1 outlining the advantages and disadvantages of running an application on Raspberry Pi,

Table 4-1 - Advantages and Disadvantages Running an Application in Raspberry Pi

<b>Advantages</b>	<b>Disadvantages</b>
Cost-effective solution	Limited processing power compared to high-end systems
Wide availability and community support	Limited memory capacity and storage options
Versatile and customizable platform	Limited GPU capabilities for intensive computations
Built-in connectivity options (Ethernet, Wi-Fi)	Limited support for demanding or resource-intensive tasks
Support for various operating systems and software tools	Limited I/O ports for extensive sensor or peripheral usage
Hardware acceleration for image and video processing	Limited scalability for large-scale deployments
Official camera modules for easy image capture	Relatively small form factor may limit expansion capabilities
Power-efficient operation	Reliance on external power supply for prolonged usage

It's important to consider the specific requirements of this image processing application and the trade-offs associated with running it on Raspberry Pi. While Raspberry Pi offers numerous advantages such as cost-effectiveness, community support, and connectivity options, it has limitations in terms of processing power, memory capacity, and scalability. These limitations may impact the performance and capability of resource-intensive or large-scale applications. However, for small to medium-sized projects or prototypes, Raspberry Pi can provide a versatile and affordable platform for running various types of applications, including image processing tasks.

#### 4.1.1 Hardware Specifications of Specifications Raspberry Pi

The Table 4-2 demonstrate the specification of the Raspberry Pi computer. [18]

Table 4-2 – Hardware Specifications of Raspberry Pi

<b>Processor</b>	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
<b>Memory</b>	4GB LPDDR4 SDRAM
<b>Connectivity</b>	2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE Gigabit Ethernet 2 × USB 3.0 ports 2 × USB 2.0 ports.
<b>GPIO</b>	Standard 40-pin GPIO header (fully backwards-compatible with previous boards)
<b>Video &amp; Sound</b>	2 × micro HDMI ports (up to 4Kp60 supported) 2-lane MIPI DSI display port

	2-lane MIPI CSI camera port 4-pole stereo audio and composite video port
<b>Multimedia</b>	H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES, 3.0 graphics
<b>SD card support</b>	Micro SD card slot for loading operating system and data storage
<b>Input Power</b>	5V DC via USB-C connector (minimum 3A) 5V DC via GPIO header (minimum 3A) Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
<b>Production lifetime</b>	The Raspberry Pi 4 Model B will remain in production until at least January 2026.

Some alternative platforms offer better scalability for large-scale deployments compared to Raspberry Pi. Here are a few examples.

1. **NVIDIA Jetson Series:** The NVIDIA Jetson series, such as Jetson Nano, Jetson Xavier NX, and Jetson AGX Xavier, are powerful edge computing platforms specifically designed for AI and computer vision applications. They feature high-performance GPUs, multiple CPU cores, and hardware accelerators. Jetson devices are suitable for demanding image processing tasks and offer better scalability for large-scale deployments.
2. **Intel NUC (Next Unit of Computing):** Intel NUCs (Next Unit of Computing) are compact, yet powerful, mini-PCs that offer a range of CPU and GPU options. They provide more processing power and memory capacity than Raspberry Pi, making them suitable for image processing and other computationally intensive applications. Intel NUCs can be scaled up by deploying multiple units in a cluster or distributed computing setup.
3. **AWS IoT Greengrass:** AWS IoT Greengrass is a cloud based IoT platform by Amazon Web Services (AWS). It extends cloud capabilities to the edge devices, allowing you to perform local data processing and analytics. AWS IoT Greengrass enables large-scale deployments by leveraging cloud scalability and management features while distributing processing tasks to edge devices.
4. **Google Cloud IoT Edge:** Google Cloud IoT Edge is a similar solution to AWS IoT Greengrass, providing cloud capabilities at the edge. It allows you to run IoT applications on edge devices and seamlessly integrate with Google Cloud services. Google Cloud IoT Edge offers scalability and management features for large-scale deployments.
5. **Industrial-Grade Single Board Computers (SBCs):** There are industrial-grade SBCs available in the market that offer enhanced durability, reliability, and scalability compared to consumer-level boards like Raspberry Pi. These SBCs are designed for industrial IoT applications and can handle more demanding processing requirements and rugged environments. [17]

When considering large-scale deployments, it's essential to evaluate the application's specific requirements, including processing power, memory, scalability, and management capabilities. Factors such as cost, compatibility with your software stack, and support from the platform's community or ecosystem should also be considered. However, Raspberry Pi was used in the project as a compatible solution.

## 4.2 IM7600G-H 4G HAT (B)

The IM7600G-H 4G HAT (B) is an add-on module designed to provide cellular connectivity to Raspberry Pi, enabling internet connection for IoT applications in scenarios where traditional Wi-Fi or Ethernet connectivity may not be available or practical. [19]. The below Figure 4-1 - IM7600G-H 4G HAT (B) is a suitable device for this project.



Figure 4-1 - IM7600G-H 4G HAT (B)

Here are some reasons that need the IM7600G-H 4G HAT (B) for this IoT project:

1. **Remote Connectivity:** The 4G HAT allows your Raspberry Pi to connect to the internet via cellular networks, providing remote connectivity without relying on local Wi-Fi networks or wired Ethernet connections. This is especially useful in remote locations, outdoor environments, or mobile applications where Wi-Fi coverage may be limited or unavailable.
2. **Flexibility and Mobility:** With the 4G HAT, your Raspberry Pi becomes mobile and can be easily deployed in different locations without the need for a fixed internet connection. This flexibility is advantageous for applications such as asset tracking, mobile monitoring systems, or IoT deployments in vehicles or drones.
3. **Backup or Redundant Connectivity:** The 4G HAT can serve as a backup or redundant internet connection in case of Wi-Fi or Ethernet network failures. This ensures uninterrupted connectivity and operation of your IoT application, even during network outages.
4. **Wide Network Coverage:** Cellular networks provide extensive coverage in many regions, making the 4G HAT suitable for IoT applications that require connectivity in remote or rural areas where wired or Wi-Fi connections may be limited.



5. **Secure and Encrypted Communication:** Cellular networks often employ encryption and security protocols, ensuring secure communication between your IoT devices and the internet. This is particularly important for applications that handle sensitive data or require secure data transmission.
6. **Scalability:** Cellular connectivity through the 4G HAT allows for scalability in IoT deployments. You can easily add more Raspberry Pi devices with 4G HATs to your network without the need for additional Wi-Fi infrastructure, simplifying the expansion of the IoT solution.
7. **Integration with Cloud Services:** The 4G HAT enables your Raspberry Pi to connect to cloud platforms and services, facilitating data transfer, remote management, and integration with cloud-based IoT platforms. This connectivity enables real-time data monitoring, analytics, and control of your IoT devices from anywhere.

It's important to note that the specific choice of the IM7600G-H 4G HAT (B) may depend on the chosen cellular providers region's cellular network compatibility and project's requirements. Always ensure compatibility, review documentation, and consider factors such as data plans, signal strength, and available cellular bands when selecting a cellular connectivity module for this Raspberry Pi-based IoT application.

Choosing the IM7600G-H 4G HAT (B) for Raspberry Pi offers several advantages for IoT applications requiring cellular connectivity. Here's a Table 4-3 outlining the advantages and disadvantages.

Table 4-3 - IM7600G-H 4G HAT (B) Advantages and Disadvantages

<b>Advantages</b>	<b>Disadvantages</b>
Enables remote connectivity in areas without Wi-Fi coverage	Requires a compatible cellular network and data plan
Provides flexibility and mobility for mobile IoT deployments	Additional cost for the 4G HAT module and cellular data usage
Serves as a backup or redundant internet connection	Limited data speeds compared to wired or high-speed Wi-Fi
Wide coverage area with cellular networks	Potential signal strength limitations in certain locations
Secure and encrypted communication over cellular networks	Limited compatibility with some Raspberry Pi models
Scalable for expanding IoT deployments	Additional power consumption for cellular connectivity
Integration with cloud services for remote management	Potential latency or network congestion in busy cellular networks

### 4.2.1 Features Overview

1. Connectivity Options:
  - Pogo pin and MicroUSB connector for system connection.
  - Dedicated pogo pin for Raspberry Pi Zero/Zero W.
  - MicroUSB connector for compatibility with other Raspberry Pi boards or a PC.
2. 4G Network Support:
  - Incorporates SIM7600G-H global band 4G module.
  - Compatible with 2G/3G/4G networks worldwide.
3. USB HUB Connector:
  - Allows USB extension and provides 4G network access to other Raspberry Pi boards or PCs.
4. Wide Range of Supported Functions:
  - Supports dial-up, telephone calls, SMS, TCP, UDP, DTMF, HTTP, FTP, and more.
  - Supports positioning of GPS, BeiDou, Glonass, GALILEO, QZSS, and LBS base station.
5. SIM Card Compatibility:
  - SIM card slot supports both 1.8V and 3V SIM cards.
6. Audio Capabilities:
  - Integrated onboard audio jack and audio decoder for telephone call functionality.
7. Status Monitoring:
  - Two LED indicators for monitoring the system's operating status.
8. Control and Configuration:
  - Controlled using AT commands following the 3GPP TS 27.007, 27.005, and V.25TER command set.
9. SIM Application Toolkit (SAT) Support:
  - Supports SIM application toolkit features including SAT Class 3, GSM 11.14 Release 99, and USAT.
10. Development Resources:

- Comprehensive development resources and manual provided.
- Examples tailored specifically for Raspberry Pi.

These features collectively enhance the system's connectivity, versatility, and functionality, making it suitable for integrating various applications and projects.

#### 4.2.2 Hardware Specifications of IM7600G-H 4G LTE Hat

The below Table 4-4 demonstrate the specifications of the LTE module. [19]

Table 4-4 – Hardware Specifications of IM7600G-H 4G LTE Hat

FREQUENCY BAND	
LTE Cat-4	LTE-FDD: B1/B2/B3/B4/B5/B7/B8/B12/B13/B18/B19/B20/B25/B26/B28/B66 LTE-TDD: B34/B38/B39/B40/B41
3G	UMTS/HSDPA/HSPA+: B1/B2/B4/B5/B6/B8/B19
2G	GSM/GPRS/EDGE: 850/900/1800/1900 MHz
GNSS	
Satellite systems	GPS/Beidou/GLONASS/GALILEO/QZSS
Receiver type	16-channel
	C/A Code
Sensitivity	-159 dBm (GPS) / -158 dBm (GLONASS) / TBD (BeiDou)
	Cold starts: -148 dBm
Time-To-First-Fix (open air)	Cold starts: <35s
	Hot starts: <1s
SMS AND AUDIO	
SMS	Supported types: MT, MO, CB, Text, PDU
Audio feature	Supports echo cancellation
	Supports noise reduction
OTHER	
Power supply	5V
Operating temperature	-30°C ~ 80°C
Storage temperature	-45°C ~ 90°C
Dimensions	65.00 × 32.00mm

## **4.3 Power Supply**

For several reasons, the power supply is of utmost importance in an image-processing IoT traffic management system. Firstly, the system relies on continuous power to operate its hardware components, such as cameras, processors, and communication modules.

Interruptions in power supply can lead to system shutdowns, resulting in a loss of real-time monitoring and traffic management capabilities. Power outages can disrupt data transmission, causing delays or loss of critical information. A reliable power source also ensures the system's availability and functionality, enabling it to operate effectively and provide accurate traffic analysis and management. Therefore, a stable and uninterrupted power supply is crucial for the system's reliability, performance, and overall success.

### **4.3.1 Calculating the Power Requirements**

To calculate the power requirements for your Raspberry Pi with a GSM module, we need to consider the power consumption of both devices. Here's a general approach.

### **4.3.2 Raspberry Pi Power Consumption**

The power consumption of a Raspberry Pi varies depending on the model and the tasks it's performing. As a rough estimate, let's assume it consumes around 2.5 Watts on average. Remember that power consumption may increase if you're using peripherals or running resource-intensive applications.

### **4.3.3 GSM Module Power Consumption**

Depending on the specific model and usage, GSM modules also have varying power consumption. You'll need to refer to the datasheet or technical specifications of your GSM module to determine its power consumption. Typically, GSM modules consume around 0.5-2 Watts during active communication.

### **4.3.4 Total Power Consumption**

Assuming the Raspberry Pi consumes 3 W, and the GSM module consumes 1.5 Watt (as an average estimate), the total power consumption would be around 4.5 W. Consider it 5W.

### **4.3.5 Battery Capacity Calculation**

The desired operating time and the battery's capacity must be considered to determine the suitable battery pack. As the battery should have the capacity to work in the daytime, an approximate calculation of 16 hours is considered. The reason is that at nighttime, the polls will get power, and the battery pack can be powered up. Therefore, a pack that can provide enough power for that duration must be used.

**Battery Capacity (in Wh) = Total Power Consumption (Watts) \* Time (in hours)**

**Battery Capacity = 5 Watts \* 16 hours = 80 Watt-hours**

It's important to note that this calculation assumes a continuous power draw without any power-saving measures. In practice, your actual battery life may vary based on factors such as power management settings, idle time, and the real power consumption of your specific devices.

Selecting a Suitable Battery Pack: The battery setup below is suitable based on the calculated battery capacity of 80 Watts-hours.

**1. TalentCell 12V Lithium-ion Battery PB120B2, Rec PB120B2**

The Table 4-5 demonstrate the technical details of the power supply. This device is a suitable option for power backups when considering the power requirement factors. [20]

Table 4-5 – Technical Specifications of TalentCell 12V Lithium-ion Battery PB120B2

<b>Model</b>	PB120B2	<b>Capacity</b>	3.7V 38400mAh / 142.08Wh
<b>Input</b>	12.6V/2A	<b>Weight</b>	About 730g
<b>Output</b>	12V (voltage range is 12.6-9V) /6A Max. 5V/2.4A Max		



Figure 4-2 - TalentCell 12V Lithium-ion Battery PB120B2

## 2. Anker 737 Powerbank 140 W PD 3.1 24000 mAh

Powerful power bank for laptops and other devices. It has the facility of, [21]

- Integrated status display
- Powerful enough for laptops
- High capacity of 24,000 mAh

Anker 737 is a powerful power bank with USB PD 3.1 for laptops, tablets, mobile phones and other accessories. Equipped with an integrated status display that shows the remaining battery time and how much the individual ports deliver. High capacity of 24,000 mAh and powerful enough for laptops with up to 140 W PD charging.



Figure 4-3 - Anker 737 Powerbank 140 W PD 3.1 24000 mAh

## 4.4 Camera Module

The Microsoft LifeCam Cinema is a webcam for business with high-quality 720p HD widescreen video and clear audio. It has the below functionalities.

- Records true HD-quality video at up to 30 fps
- Autofocus functionality
- High-precision glass element lens
- Digital microphone



Figure 4-4 - Microsoft LifeCam Cinema

The below Table 4-6 shows the specifications of the Microsoft LifeCam Cinema which is suitable for the device.

Table 4-6 - Microsoft LifeCam Cinema Specification

<b>Attribute</b>	<b>Value</b>
Maximum Resolution	1280 x 720
Model	LifeCam Cinema
Built-In Microphone	Yes
Connection Type	USB 2.0
Effective Pixels	5MP
Frame Rate	30fps
Attribute	Value
Maximum Resolution	1280 x 720
Model	LifeCam Cinema
Built-In Microphone	Yes
Connection Type	USB 2.0
Effective Pixels	5MP
Frame Rate	30fps

## 4.5 Integrated Unit

The below Figure 4-5 showcases the configuration of the edge computer specifically designed for the IoT unit.

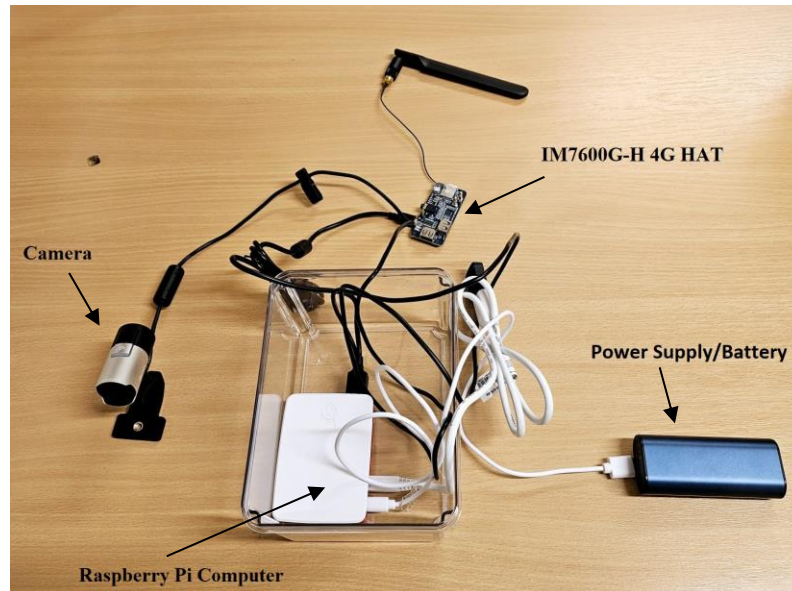


Figure 4-5 - Integrated Unit that Used for the Testing Purpose

The integrated unit that was developed serves as a prototype for demonstrating our innovative traffic management IoT project. The system comprises various components working together seamlessly. Cameras are deployed to capture the video feed of the traffic scenario. This video feed is then transmitted to a Raspberry Pi, which acts as the processing unit for image recognition. The Raspberry Pi employs advanced algorithms and machine learning techniques to analyze the video frames and extract relevant traffic information, such as vehicle detection, traffic flow, and congestion levels.

Once the data is processed and transmitted to the cloud using an LTE module connection. This enables real-time data transfer and ensures that the information is readily available for further analysis and decision-making. The cloud-based platform facilitates storing, managing, and analyzing large volumes of traffic data, providing valuable insights into traffic patterns, optimizing signal timings, and improving overall traffic management strategies. One critical aspect of the system is the power supply. A reliable and robust power supply solution is implemented to ensure continuous operation, even in the absence of power connectivity throughout the day. This may involve using battery banks or alternative power sources to sustain the system's power requirements for extended periods.

These compact IoT units find practical application in various traffic management scenarios, including parking systems and urban traffic control. By leveraging the power of image processing, real-time data analysis, and cloud computing, our traffic management IoT system aims to enhance efficiency, reduce congestion, and improve overall transportation experiences for both commuters and authorities.



# 5 Configuring Edge Microprocessor

To enable the smooth execution of firmware on the microprocessor in the image recognition IoT project, it is necessary to configure the dependencies and libraries on the Raspberry Pi. This ensures that the necessary tools and resources are in place for efficient image recognition operations. By setting up the dependencies and libraries, the project can leverage the full capabilities of the Raspberry Pi, enabling accurate and reliable image recognition and enhancing the system's overall functionality. The setup is as follows.

## 5.1 Edge computer – Raspberry Pi setup

Installing the listed dependencies on a Raspberry Pi is essential for running YOLOv5, a popular object detection algorithm. By installing these dependencies, you ensure that the necessary software tools and libraries are present on the Raspberry Pi to support the execution of YOLOv5 effectively. The dependencies include packages for image processing, deep learning, data visualization, exporting models to different formats, and deployment. Running YOLOv5 on a Raspberry Pi allows for real-time object detection and localization on a compact and energy-efficient device. This is particularly important for applications where deploying a powerful server or cloud-based solution is not feasible or practical. The Raspberry Pi's portability, low power consumption, and affordability make it an attractive choice for edge computing scenarios, such as robotics, home automation, surveillance systems, and IoT devices.

The dependencies play crucial roles in enabling YOLOv5's functionality and performance on the Raspberry Pi. Packages like `numpy`, `opencv-python`, and `pillow` provide efficient image processing capabilities, allowing YOLOv5 to analyze and detect objects in images or video streams. `Torch` and `torch-vision` provide the core deep learning framework, enabling the training and inference of YOLOv5 models. Other dependencies, such as `matplotlib`, `pandas`, and `seaborn`, facilitate data visualization and analysis, aiding in model evaluation and result interpretation.

Furthermore, the export-related dependencies, like `coremltools`, `onnx`, and `tensorflow`, enable the conversion and deployment of YOLOv5 models to various formats and platforms. This flexibility allows for seamless integration with different target environments, such as mobile devices, web applications, and specialized hardware accelerators.

In summary, installing the listed dependencies on a Raspberry Pi is vital to harness the power of YOLOv5 for real-time object detection and localization on a compact and energy-efficient edge computing device. These dependencies provide the necessary tools, libraries, and export capabilities to ensure a smooth and efficient deployment of YOLOv5 on the Raspberry Pi, enabling a wide range of robotics, home automation, surveillance, and IoT applications.

### 5.1.1 Making Virtual Environment to Install Dependencies.

Using a virtual environment for installing certain dependencies is a good practice in Python development, including when working with YOLOv5 on a Raspberry Pi. Virtual environments provide an isolated and controlled environment for installing and managing Python packages specific to a project, separate from the system-level Python installation.

Here are the reasons why virtual environments are beneficial when installing dependencies:

1. **Dependency Isolation:** Virtual environments allow you to create an isolated environment for each project, ensuring that the installed dependencies do not conflict with each other or with the system-level packages. This helps avoid version conflicts and lets you maintain different versions of the same package across different projects.
2. **Reproducible Environments:** By creating a virtual environment, you can explicitly manage and document the exact versions of the dependencies required for your project. This ensures that the project can be easily reproduced on different machines or at different times, providing consistency and reproducibility.
3. **Easy Setup and Cleanup:** Virtual environments provide a straightforward setup process, allowing you to create and activate a dedicated environment for your project. Additionally, if you decide to remove or clean up the project, you can simply delete the virtual environment, eliminating any installed packages and dependencies associated with the project.
4. **Security and Stability:** By using a virtual environment, you reduce the risk of inadvertently modifying or interfering with system-level packages critical for the stability and security of your Raspberry Pi. This provides an extra layer of protection and helps maintain a stable and secure system.
5. **Collaboration and Sharing:** Virtual environments make it easier to collaborate on projects with others. By sharing the project's virtual environment configuration, including the list of dependencies, collaborators can quickly set up the same environment and work with the project seamlessly.

Overall, virtual environments offer a clean and isolated environment for installing dependencies, ensuring dependency management, reproducibility, ease of setup, security, and collaboration. Using virtual environments when installing YOLOv5's dependencies on a Raspberry Pi helps create a controlled and self-contained environment specific to the project, ensuring smooth and reliable execution of the object detection algorithm.

### 5.1.2 List of Dependencies

To install YOLOv5 on a Raspberry Pi, you would need to install the following dependencies listed below

Base:

- `gitpython>=3.1.30`

- matplotlib>=3.3
- numpy>=1.23.5
- opencv-python>=4.1.1
- pillow>=10.3.0
- psutil
- PyYAML>=5.3.1
- requests>=2.23.0
- scipy>=1.4.1
- thop>=0.1.1
- torch>=1.8.0
- torchvision>=0.9.0
- tqdm>=4.64.0
- ultralytics>=8.0.232

#### Logging:

- tensorboard (optional)
- clearml (optional)
- comet (optional)

#### Plotting:

- pandas>=1.1.4
- seaborn>=0.11.0

#### Export:

- coremltools (optional)
- onnx (optional)
- onnx-simplifier (optional)
- nvidia-pyindex (optional)
- nvidia-tensorrt (optional)
- scikit-learn (optional)
- tensorflow (optional)

- tensorflowjs (optional)
- openvino-dev (optional)

Deploy:

- setuptools>=65.5.1
- tritonclient[all]~=2.24.0 (optional)

Extras:

- ipython (optional)
- mss (optional)
- albumentations>=1.0.3 (optional)
- pycocotools>=2.0.6 (optional)
- wheel>=0.38.0

Please note that not all dependencies are required for every use case, and some are optional for specific functionalities or export formats.

# 6 Server-Side Resources Implementation

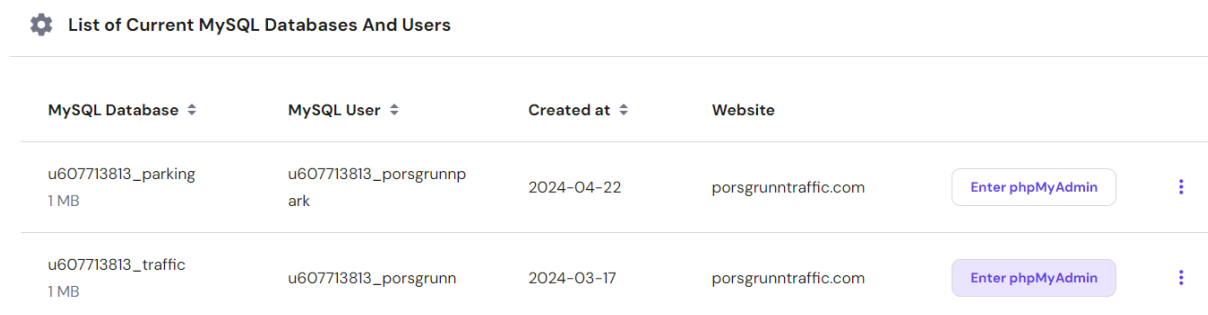
The implementation of an IoT image recognition traffic management and vehicle parking system requires the utilization of a cloud-based database and an online data analysis tool. These components are essential for effectively uploading and analyzing the collected data. By leveraging a cloud database and an online data analysis tool, the system ensures efficient storage, processing, and visualization of real-time traffic and parking data, enabling informed decision-making, and enhancing overall system performance.

## 6.1 MySQL Database Implementation

This master thesis involves building a database system using PHP and MySQL with the assistance of phpMyAdmin. PHP is a server-side scripting language that allows for dynamic web development, while MySQL is a popular open-source relational database management system.

A robust and efficient database is constructed to store and manage several data types using PHP and MySQL. The database design includes creating tables, defining fields, and establishing relationships between different entities. This allows for structured and organized data storage. PhpMyAdmin, a web-based database management tool, facilitates the administration and maintenance of the MySQL database. It provides a user-friendly interface for tasks such as creating databases, executing SQL queries, managing tables and records, and configuring database settings.

The database built using PHP and MySQL can support many applications, including storing user information, managing product catalogs, handling transactional data, and more. The flexibility and scalability of PHP and MySQL make them suitable for handling large datasets and accommodating future growth. By utilizing PHP and MySQL in conjunction with phpMyAdmin, this master thesis aims to create a reliable and efficient database system that can effectively store and retrieve data for various applications. The combination of these technologies allows for seamless integration with web-based interfaces and provides a solid foundation for data-driven decision-making and analysis. The Figure 6-1 - MySQL shows the current users and Databases in web.



The screenshot shows the phpMyAdmin interface with the title "List of Current MySQL Databases And Users". It displays a table with four columns: "MySQL Database", "MySQL User", "Created at", and "Website". There are two rows of data, each with a "Enter phpMyAdmin" button and a vertical ellipsis menu icon to its right.

MySQL Database ↕	MySQL User ↕	Created at ↕	Website	
u607713813_parking 1 MB	u607713813_porsgrunnp ark	2024-04-22	porsgrunntraffic.com	<a href="#">Enter phpMyAdmin</a> ⋮
u607713813_traffic 1 MB	u607713813_porsgrunn	2024-03-17	porsgrunntraffic.com	<a href="#">Enter phpMyAdmin</a> ⋮

Figure 6-1 - MySQL Databases

### 6.1.1 Database for Parking Vehicles.

The ER diagram in Figure 6-2 shows a relational database design for parking lot management. It consists of three tables: `parking_data`, `parking_location`, and `empty_areas`.

- **parking\_data table:** This table likely stores overall parking lot usage data. It has the following attributes:
  - `id`: An integer that likely serves as the primary key for the table.
  - `car_count`: An integer that stores the total number of cars currently parked in the lot.
  - `free_space`: An integer that stores the number of free parking spaces currently available.
  - `time`: A datetime value that stores the date and time the data was collected.
- **parking\_location table:** This table likely stores data for specific parking locations within the lot. It has the following attributes:
  - `ID`: An integer that likely serves as the primary key for the table.
  - `place`: A varchar attribute that might store a location identifier for a specific parking spot
  - `capacity`: An integer that stores the total capacity of the parking location, possibly the number of vehicles it can accommodate.
- **empty\_areas table:** This table's purpose is less clear from the limited attributes shown. It has the following attributes:
  - `id`: An integer that likely serves as the primary key for the table.
  - `area_name`: A varchar attribute that stores the free slots of a parking area.
  - `time`: A varchar attribute that stores a time value, possibly indicating the last time the area was checked for emptiness.
  - `place`: A varchar attribute that might store a location identifier for a specific parking spot.

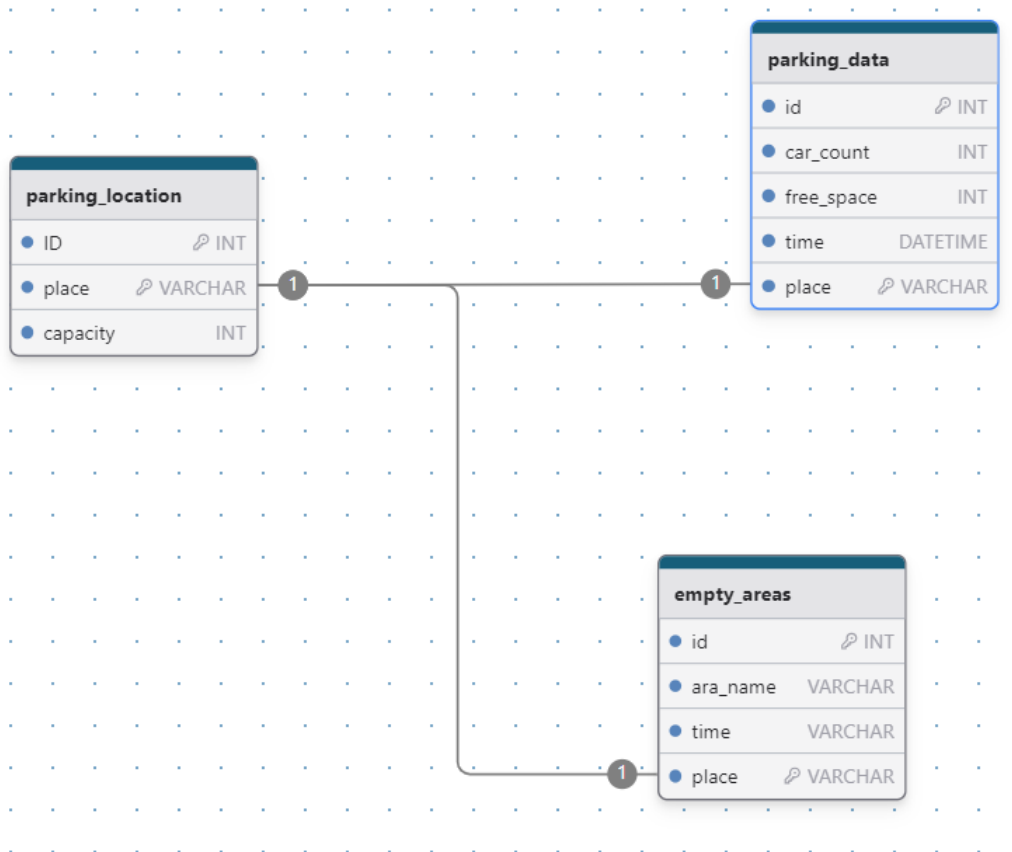


Figure 6-2 - Entity Relationship Diagram of Car Parking Database

The relationships between the tables are not explicitly shown in the ERD, but we can make some educated guesses based on the table names and attributes. Here are sample database queries for the database creation.

Columns:

- id (INT, primary key, auto-increment)
- car\_count (INT)
- free\_space (INT)
- time (DATETIME)
- place (VARCHAR or INT, depending on how you store the place information)

To create one of these tables, you can use the following SQL query:

```
CREATE TABLE parking_data (
    id INT AUTO_INCREMENT PRIMARY KEY,
```

```

car_count INT,
free_space INT,
time DATETIME,
place VARCHAR(255)
);

```

This query creates the "parking\_data" table with the specified columns: "id", "car\_count", "free\_space", "time", and "place". The "id" column is set as the primary key with auto-increment, ensuring a unique identifier for each row. The "car\_count" and "free\_space" columns are of type INT to store the count of cars and available spaces, respectively. The "time" column is of type DATETIME to store the timestamp, and the "place" column is of type VARCHAR(255) to store the place information.

### Database view in PhpMyAdmin

#### 6.1.2 The below Database for Counting the Vehicles

The schema for the "vehicle\_counts" table can be described as following **Error! Reference source not found.**

Table 6-1 - vehicle\_counts Database Structure

Field Name	Data Type	Description
timestamp	datetime	Represents the date and time when the vehicle counts were recorded.
car_count_outgoing	int	Stores the count of outgoing cars.
truck_count_outgoing	int	Stores the count of outgoing trucks.
bus_count_outgoing	int	Stores the count of outgoing buses.
free_space_outgoing	int	Represents the available free parking space for outgoing vehicles.
car_count_incoming	int	Stores the count of incoming cars.
truck_count_incoming	int	Stores the count of incoming trucks.
bus_count_incoming	int	Stores the count of incoming buses.
free_space_incoming	int	Represents the available free parking space for incoming vehicles.
place	Varchar	Represent the place of mounting the module

This schema allows for the storage of data related to vehicle counts, including the number of cars, trucks, and buses both incoming and outgoing, as well as the available free parking space for each type of vehicle. The "timestamp" field records the date and time of the data entry.



The use of appropriate data types, such as "datetime" for the timestamp and "int" for the count fields, ensures efficient storage and retrieval of the data. This schema can serve as a foundation for capturing and analyzing vehicle count data within the specified database.

The provided code snippet demonstrates the use of prepared statements and parameter binding in PHP PDO to insert data into the "vehicle\_counts" table. However, to create the table itself, you would need to execute a separate SQL query to create the table structure. Here's an example SQL query to create the "vehicle\_counts" table with the specified columns:

```
CREATE TABLE vehicle_counts (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    timestamp DATETIME,  
    car_count_outgoing INT,  
    truck_count_outgoing INT,  
    bus_count_outgoing INT,  
    free_space_outgoing INT,  
    car_count_incoming INT,  
    truck_count_incoming INT,  
    bus_count_incoming INT,  
    free_space_incoming INT,  
    place VARCHAR(255)  
);
```

## Database view in PhpMyAdmin

The below Figure 6-4 shows how its described in the web mode.

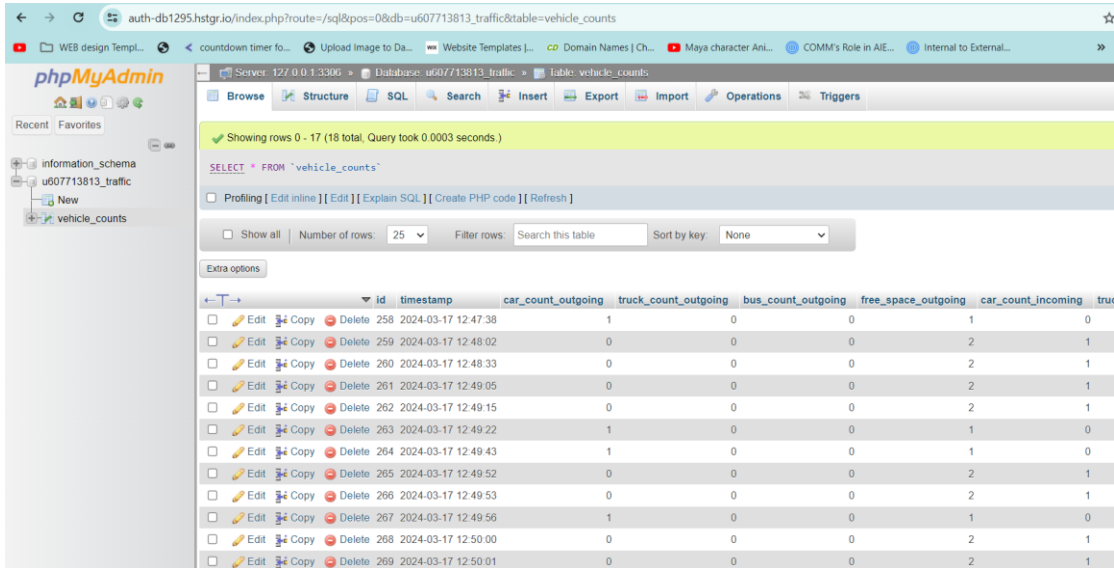


Figure 6-4 - PhpMyAdmin Database view vehical\_counts

shows how its described in the web mode.

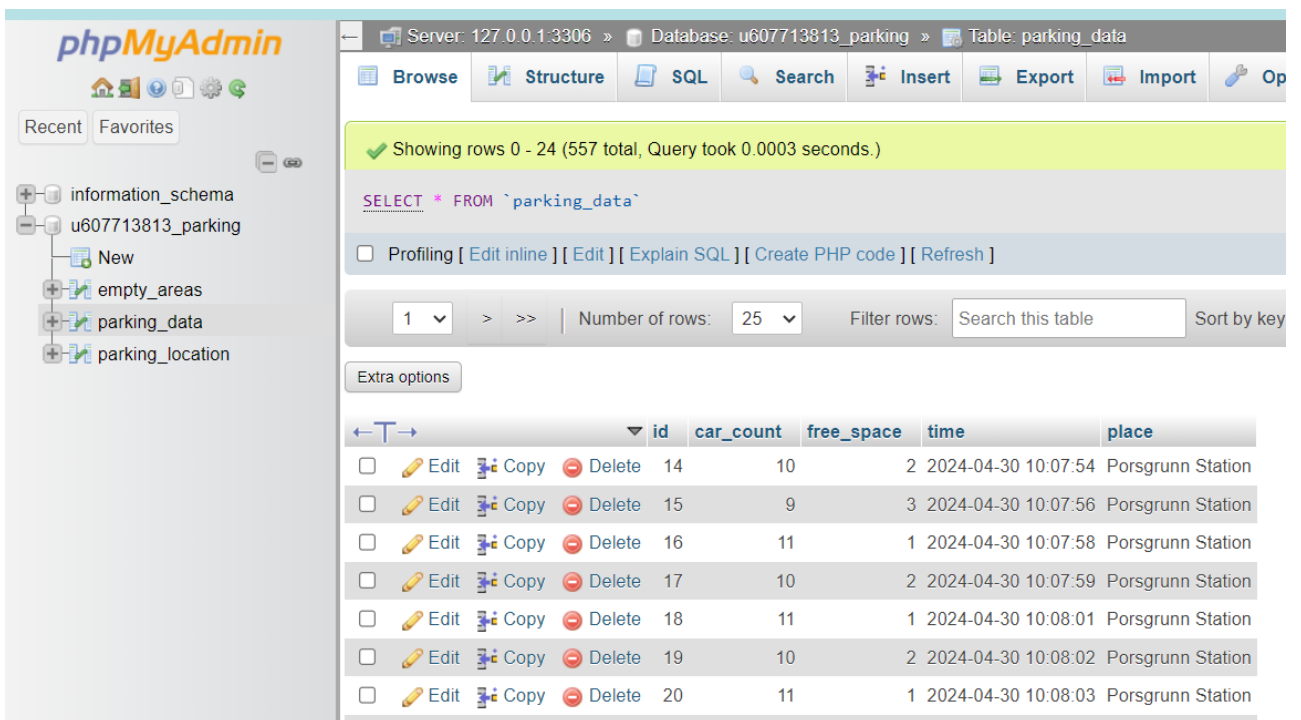


Figure 6-3 - PhpMyAdmin Database View for parking\_data

### 6.1.3 Database for Counting the Vehicles

The schema for the "vehicle\_counts" table can be described as following **Error! Reference source not found.**

Table 6-1 - vehicle\_counts Database Structure

Field Name	Data Type	Description
timestamp	datetime	Represents the date and time when the vehicle counts were recorded.
car_count_outgoing	int	Stores the count of outgoing cars.
truck_count_outgoing	int	Stores the count of outgoing trucks.
bus_count_outgoing	int	Stores the count of outgoing buses.
free_space_outgoing	int	Represents the available free parking space for outgoing vehicles.
car_count_incoming	int	Stores the count of incoming cars.
truck_count_incoming	int	Stores the count of incoming trucks.
bus_count_incoming	int	Stores the count of incoming buses.
free_space_incoming	int	Represents the available free parking space for incoming vehicles.
place	Varchar	Represent the place of mounting the module

This schema allows for the storage of data related to vehicle counts, including the number of cars, trucks, and buses both incoming and outgoing, as well as the available free parking space for each type of vehicle. The "timestamp" field records the date and time of the data entry.

The use of appropriate data types, such as "datetime" for the timestamp and "int" for the count fields, ensures efficient storage and retrieval of the data. This schema can serve as a foundation for capturing and analyzing vehicle count data within the specified database.

The provided code snippet demonstrates the use of prepared statements and parameter binding in PHP PDO to insert data into the "vehicle\_counts" table. However, to create the table itself, you would need to execute a separate SQL query to create the table structure. Here's an example SQL query to create the "vehicle\_counts" table with the specified columns:

```
CREATE TABLE vehicle_counts (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    timestamp DATETIME,  
    car_count_outgoing INT,  
    truck_count_outgoing INT,  
    bus_count_outgoing INT,
```

```

free_space_outgoing INT,
car_count_incoming INT,
truck_count_incoming INT,
bus_count_incoming INT,
free_space_incoming INT,
place VARCHAR(255)
);

```

### Database view in PhpMyAdmin

The below Figure 6-4 shows how its described in the web mode.

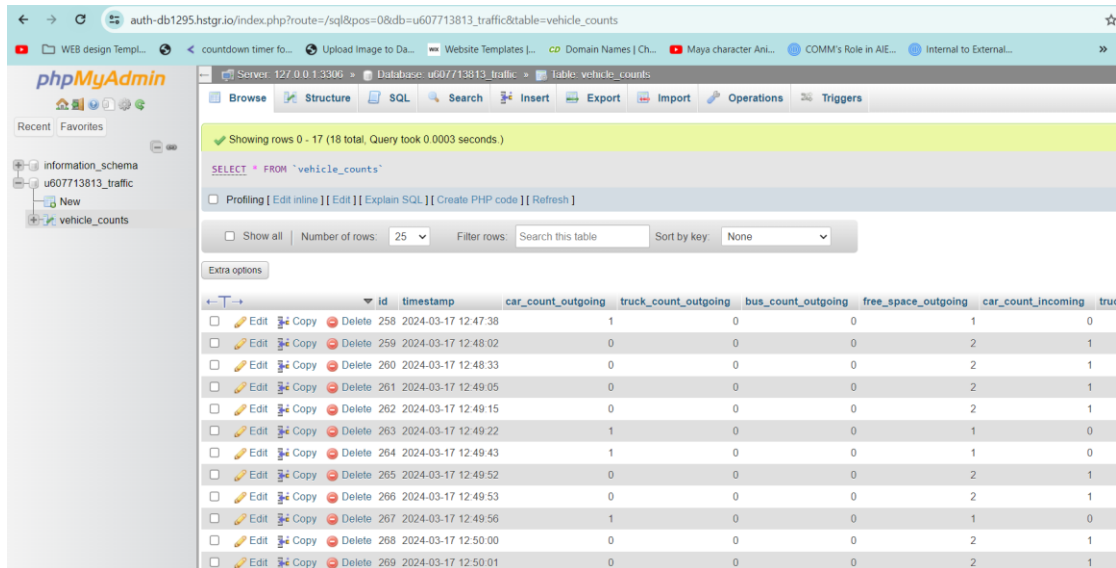


Figure 6-4 - PhpMyAdmin Database view vehical\_counts

## 6.2 Setting up the AWS Environment for Tableau Server

Tableau AWS Deployment involves seamlessly incorporating Tableau into the Cloud-Computing ecosystem of Amazon Web Services (AWS). This process entails setting up the Tableau server on a suitable infrastructure such as Amazon Elastic Compute Cloud (Amazon EC2), Microsoft Windows Server, CentOS, or Ubuntu Server. Utilizing these platforms enables users to efficiently aggregate data from various sources, empowering them to extract valuable insights by leveraging the capabilities of Tableau. Through this integration, Tableau becomes the go-to solution for analyzing and visualizing the data collected from diverse sources.

## 6.2.1 Steps to Create the Tableau Server on Cloud.

Step 1: VPC Creation for Tableau AWS Deployment involves initiating a Virtual Private Cloud (VPC) and incorporating an Amazon Elastic Network instance into it. This step guarantees a static MAC address for the Amazon EC2 instance. To accomplish this, access the Amazon VPC Console in AWS, select the desired region, and proceed to the Resources section. Initiate the VPC Wizard to configure the VPC setup. Opt for a Single Subnet configuration, assign a name to the VPC, and leave the remaining settings at their default values. The below figure Figure 6-5 - VPC Creation shows how the interface in AWS cloud

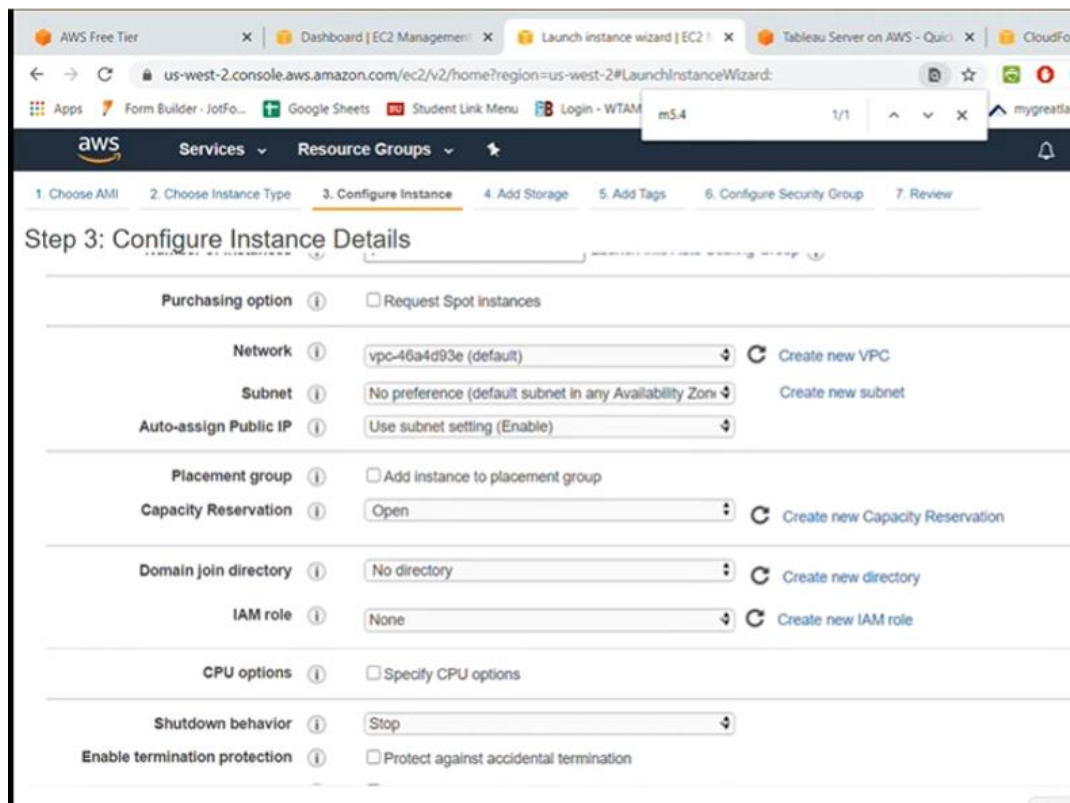


Figure 6-5 - VPC Creation for the Tableau Server

Step 2: Network and Security Configuration is the subsequent phase in deploying Tableau on AWS. Here, the aim is to restrict the incoming traffic to the VPC. Begin by selecting the appropriate Amazon EC2 instance location within the VPC using the Region Selector. Access the Navigation Pane and generate a Security Group, providing a name and description. In the Inbound tab, add rules for HTTP (port 80), HTTPS (port 443), and RDP (port 3389). By selecting "My IP" as the source, only inbound traffic from the specified IP addresses will be permitted. Alternatively, you can customize the IP range using CIDR notation. Repeat this process for each port and save the settings. The below Figure 6-6 - Network and Security Configuration demonstrates the network settings for the VPC.

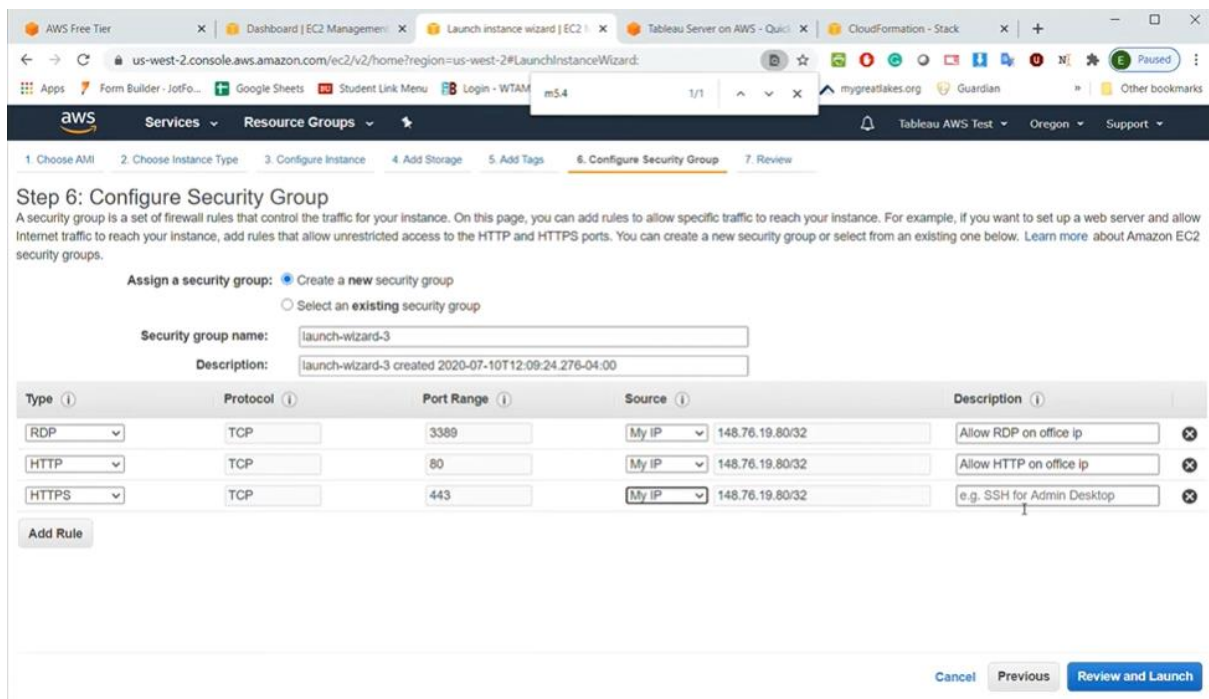


Figure 6-6 - Network and Security Configuration

Step 3: Launching an Amazon EC2 Instance is the subsequent step in deploying Tableau on AWS. Access the Amazon EC2 console and choose the designated VPC location. Under the Create Instance section, click on Launch Instance. Select an appropriate Amazon Machine Image (AMI) that meets the requirements of Tableau Server. Configure the instance details, including the Instance Type and Security Group. Review the settings and launch the Amazon EC2 instance. During the launch, create and save a new Key Pair in ".pem" format, which will be required to log onto the instance. The below Figure 6-7 shows the instance of the cloud.

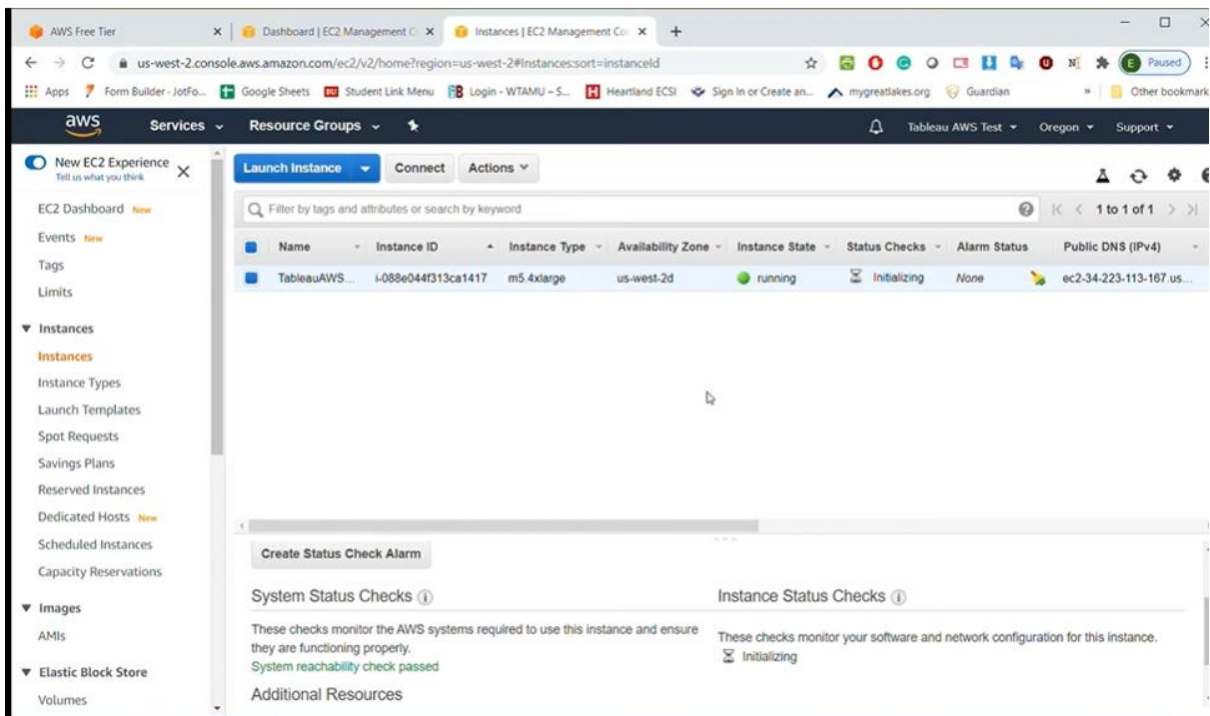


Figure 6-7 - Launching an Amazon EC2

Step 4: Creating an Elastic IP Address for the VPC is essential to allocate a static public IP address to the VPC. Access the Amazon VPC Console, choose the relevant VPC region, and navigate to Elastic IPs in the Navigation pane. Allocate a new address and associate it with the VPC instance. Copy the newly generated IP address. The below Figure 6-8 shows the IP configurations.

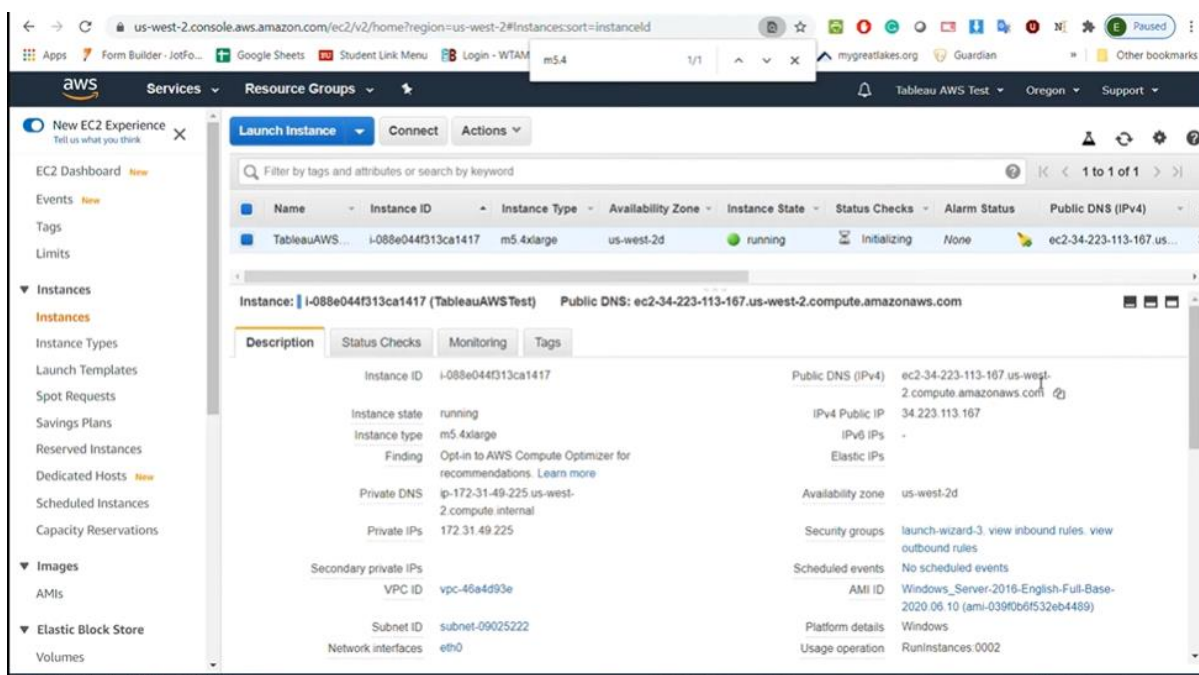


Figure 6-8 - Creating an Elastic IP Address for the VPC

Step 5: Logging onto Amazon EC2 involves installing Tableau Server on the configured EC2 instance(s). Access the Amazon VPC Console, select the appropriate region, and navigate to the EC2 Dashboard. Locate the instance and click on Connect. Download the Remote Desktop File (".rdp") to establish a connection to the instance using the previously created public IP address. Retrieve the password by providing the private key (".pem" file) from Step 3. Note the Public DNS Address, Username, and Password. Open the downloaded ".rdp" file, enter the noted credentials, and connect to the instance.

Step 6: Installing Tableau Server is the final step in Tableau AWS Deployment. Follow the standard installation procedure for Tableau Server on the Amazon EC2 Instance. Additional users can be added once installed and signed in as a Tableau Server administrator. For a cluster of Tableau Servers, repeat the installation process on other Amazon EC2 Instances. For more detailed instructions, refer to the Install and Configure section. The below figure Figure 6-9 shows the installation of the Tableau server and Figure 6-10 demonstrates how the installation of the server looks like.



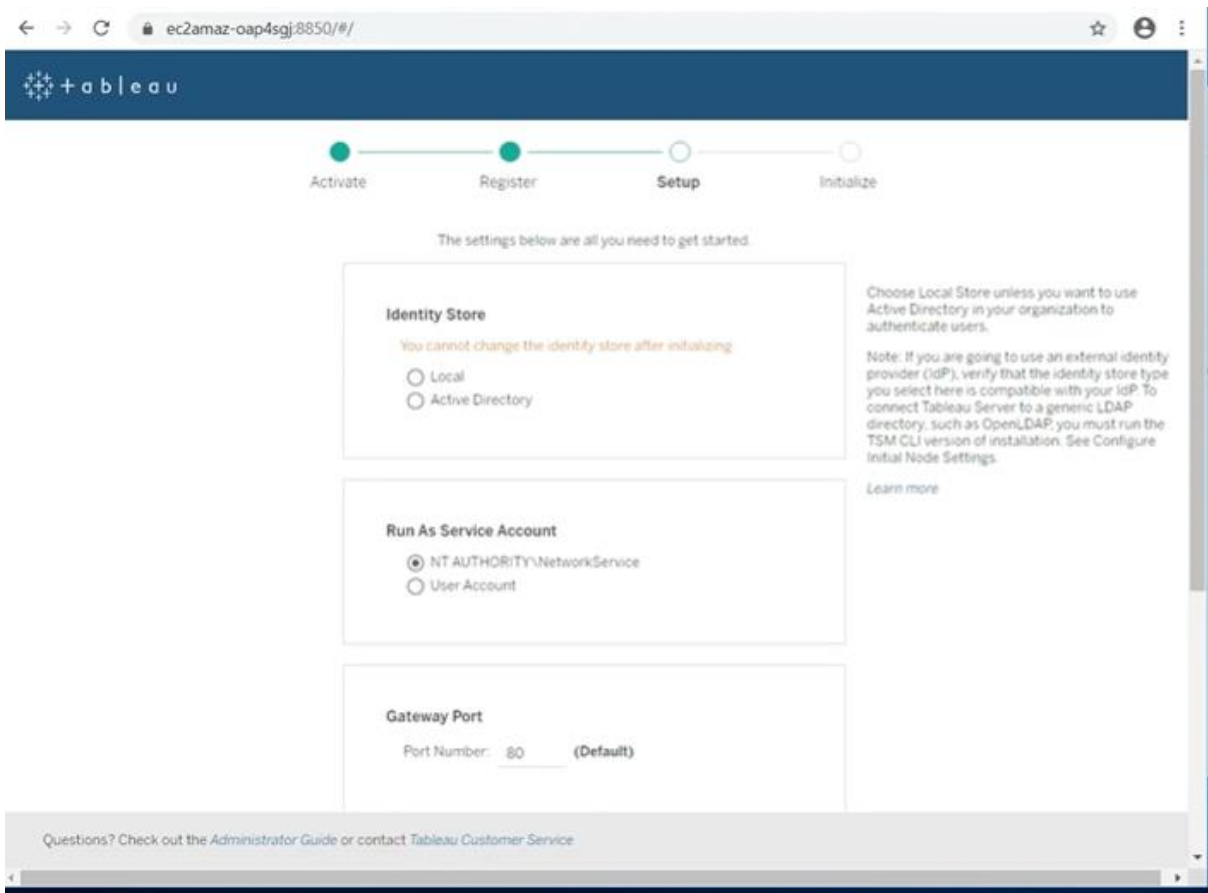


Figure 6-9 - Installing Tableau Server

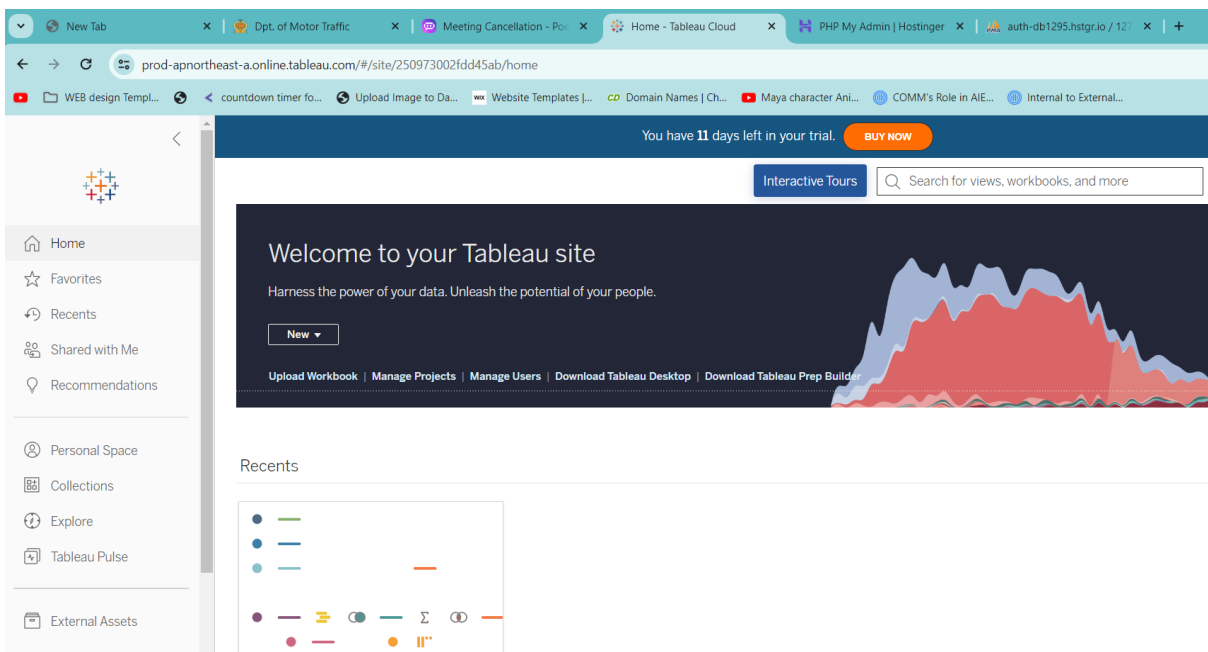


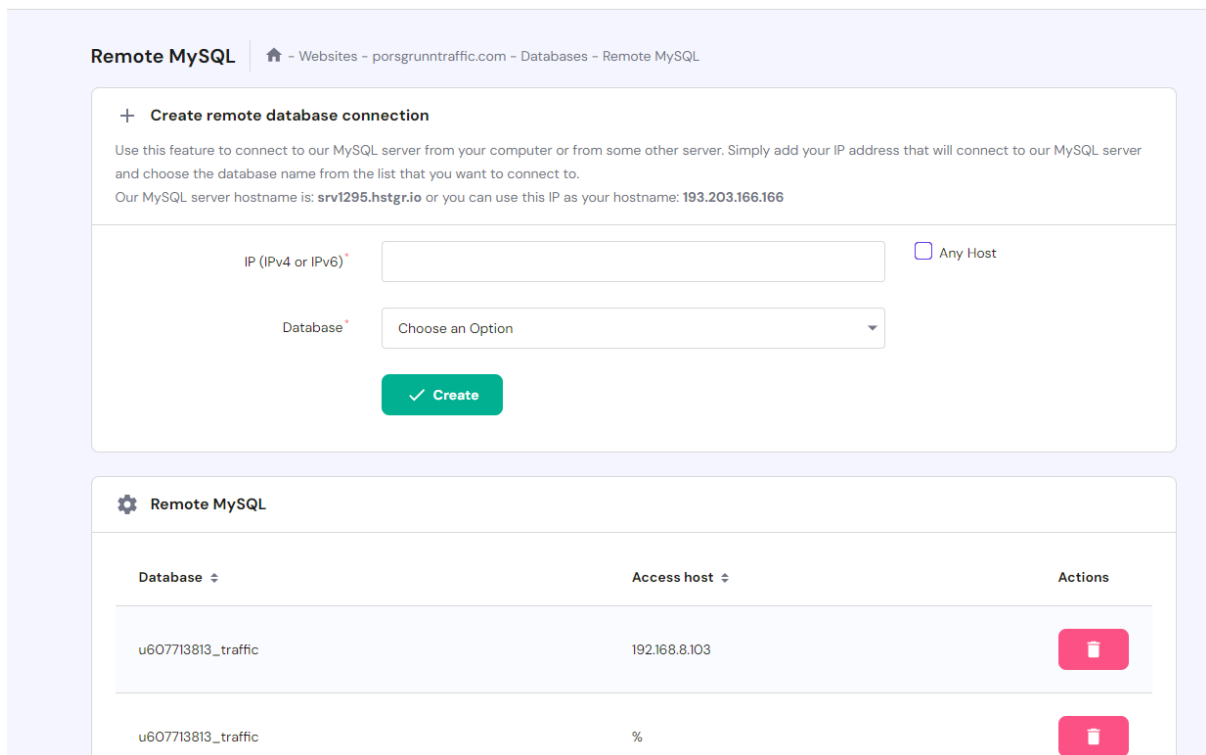
Figure 6-10 - Launching Tableau Server

## 6.2.2 Connecting the Tableau to the Database Server

To establish a connection with our MySQL server, you can utilize this convenient feature either from your computer or another server. Follow these steps to configure your connection:

1. Access the provided MySQL server hostname: `srv1295.hstgr.io`. Alternatively, you can use the IP address `193.203.166.166` as your hostname.
2. Add your specific IP address to the configuration. This will allow your IP to connect to our MySQL server securely.
3. Select the desired database from the provided list. This will determine the specific database you intend to establish a connection with.

Following these instructions, you can effortlessly connect to our MySQL server, enabling seamless data access and management. The below Figure 6-11 demonstrates the view after granting access to the MySQL server.



The screenshot shows the 'Remote MySQL' configuration page. At the top, there is a breadcrumb trail: 'Remote MySQL | Home - Websites - porsgrunntraffic.com - Databases - Remote MySQL'. Below this is a section titled '+ Create remote database connection'. The instructions state: 'Use this feature to connect to our MySQL server from your computer or from some other server. Simply add your IP address that will connect to our MySQL server and choose the database name from the list that you want to connect to. Our MySQL server hostname is: `srv1295.hstgr.io` or you can use this IP as your hostname: `193.203.166.166`'. The form includes an input field for 'IP (IPv4 or IPv6)\*', a checkbox for 'Any Host', and a dropdown menu for 'Database\*' with the text 'Choose an Option'. A green 'Create' button is at the bottom of the form. Below the form is a table titled 'Remote MySQL' with columns 'Database', 'Access host', and 'Actions'. The table contains two entries:

Database	Access host	Actions
u607713813_traffic	192.168.8.103	[Delete]
u607713813_traffic	%	[Delete]

Figure 6-11 - Granting Access to the Database in the Server.

From Tableau server or desktop, you can enter the database as below Figure 6-12.

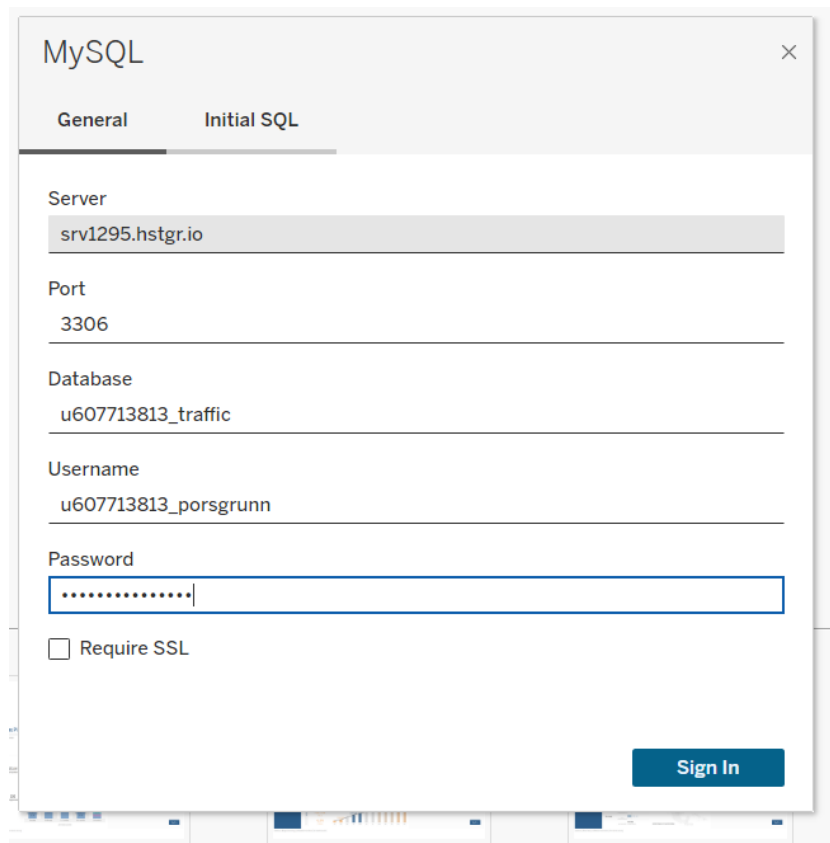


Figure 6-12 - Signing to Database via Tableau

# 7 Implementing Vehicle Counting System

The Vehicle Counting System implemented in Porsgrunn kommune utilizes cutting-edge image recognition technology to accurately track and count vehicles passing through various locations within the municipality. This advanced system is crucial in traffic management, urban planning, and infrastructure development by providing precise data and insights about vehicular movement patterns. The primary purpose of the Vehicle Counting System based on image recognition is to collect reliable and real-time information about traffic flow, vehicle volume, and related metrics. This data enables Porsgrunn kommune to make informed decisions regarding road network optimization, traffic signal timing, transportation planning, and infrastructure upgrades. By understanding traffic patterns and trends, the municipality can enhance road safety, reduce congestion, and improve the overall transportation experience for residents and visitors. The Vehicle Counting System employs state-of-the-art image recognition algorithms and cameras strategically placed at key locations throughout Porsgrunn kommune. These locations are carefully selected based on traffic density, historical data, and specific data collection goals. The cameras capture real-time video footage of the roadways, and the image recognition algorithms analyze the footage to detect and count vehicles accurately. Using computer vision techniques, the system identifies vehicles in the captured images, extracts relevant information such as vehicle type, speed, direction, and timestamp, and generates real-time data on traffic patterns. This data is then processed and analyzed to provide valuable insights into traffic flow, peak hours, congestion hotspots, and other relevant traffic metrics.

## 7.1 Data Analysis and Integration

The Vehicle Counting System based on image recognition generates large volumes of data, which is processed and analyzed using advanced algorithms and statistical techniques. By analyzing this data, traffic engineers and urban planners gain valuable insights into traffic patterns, congestion levels, and the utilization of road networks. Moreover, the Vehicle Counting System can be seamlessly integrated with other existing transportation systems and databases within Porsgrunn kommune. This integration allows for a comprehensive analysis of the transportation ecosystem by combining data from various sources, such as public transport systems, pedestrian movement, and traffic signal timings. The holistic view of the transportation network enables data-driven decision-making processes, leading to more effective traffic management strategies and infrastructure planning.

## 7.2 Privacy and Data Security

Porsgrunn kommune prioritizes individual privacy and data security when implementing the Vehicle Counting System. The image recognition technology employed ensures that no personally identifiable information is collected or stored. The system is designed to process and analyze aggregated and anonymized data, adhering to relevant privacy regulations and local laws. This approach ensures the confidentiality and security of the collected

information. In conclusion, the Vehicle Counting System utilizing image recognition technology in Porsgrunn Kommune is an advanced solution that provides accurate and valuable data on vehicular movement. By leveraging this data, the municipality can make informed decisions to improve traffic management, optimize infrastructure, and enhance the overall transportation experience for residents and visitors while maintaining privacy and data security.

## 7.3 Marking the Entry Location Points

To facilitate accurate vehicle counting, we have implemented a system that allows for the identification of custom marked locations. These marked areas serve as reference points for our image recognition technology, enabling precise and reliable vehicle counting. By leveraging the power of image recognition algorithms, we can effectively analyze the captured images and extract valuable data for this purpose.

### 7.3.1 Section 1: Importing Libraries and Initialization

This section begins by importing the necessary libraries: cv2 for computer vision tasks, numpy for numerical operations, cvzone for additional drawing capabilities, and pickle for object serialization. It then sets up the initial configuration by creating a VideoCapture object named cap to read frames from the camera (using the default camera index 0). Additionally, it initializes variables such as drawing and area\_names with their initial values.

#### Python Code section -

```
import Libraries ///
```

```
cap = cv2.VideoCapture(1)
```

```
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
```

```
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
```

```
drawing = False
```

```
area_names=[]
```

### 7.3.2 Section 2: Loading Pre-defined Polygons and Area Names

In this section, the code attempts to load pre-defined polygons and area names from a file named "freedomtech" using pickle. It handles potential errors, such as file not found or loading failure, using a try-except block. If the loading is successful, the retrieved values are assigned to the polygons and area\_names variables. However, if the loading fails or the file is not found, polygons is set as an empty list.

### Python Code section -

```
try:
    with open("freedomtech", "rb") as f:
        data = pickle.load(f)
        polylines, area_names = data['polylines'],
data['area_names']
except:
    polylines = []
```

### 7.3.3 Section 3: Mouse Event Handling for Drawing Polylines

This section defines the draw function, which acts as the event handler for mouse actions. When the left mouse button is clicked (LBUTTONDOWN), the function collects points for drawing polylines. As the mouse moves (MOUSEMOVE), it appends the current point to the points list if drawing is in progress. Upon releasing the left mouse button (LBUTTONUP), it sets the drawing flag to False, prompts the user to enter an area name using the input function, and adds the area name to area\_names if a non-empty name is provided. Finally, the collected points are converted into a NumPy array and appended to the polylines list. This is the separation section of the lanes. The incoming and outgoing. The lanes should be marked clearly showing their left and right sides.

### Python Code section -

```
points = []
current_name = ""
def draw(event, x, y, flags, param):
    global points, drawing
    drawing = True
    if event == cv2.EVENT_LBUTTONDOWN:
        points = [(x, y)]
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing:
            points.append((x, y))
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
```

```

current_name = input('areaname:-')
if current_name:
    area_names.append(current_name)
polylines.append(np.array(points, np.int32))

```

### 7.3.4 Section 4: Drawing Polylines and Displaying Area Names

This section enters a loop that continuously reads frames from the camera. Each frame is resized to a width of 1020 pixels and a height of 500 pixels. Within the loop, the code iterates over the pre-defined polylines and area\_names. For each polyline, it draws the shape on the frame using cv2.polylines and displays the corresponding area name using cvzone.putTextRect. The modified frame is then shown in a window titled 'FRAME' using cv2.imshow. The mouse callback function is set to draw using cv2.setMouseCallback to enable drawing new polylines and capturing area names. The loop waits for a key press with a 100ms delay, and if the 's' key is pressed, it saves the polylines and area\_names to the "freedomtech" file using pickle.

#### Python Code section -

```

while True:
    ret, frame = cap.read()
    if not ret:
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
        continue
    frame = cv2.resize(frame, (1020, 500))
    for i, polyline in enumerate(polylines):
        cv2.polylines(frame, [polyline], True, (0, 0,
255), 2)
        cvzone.putTextRect(frame, f'{area_names[i]}',
tuple(polyline[0]), 1, 1)
    cv2.imshow('FRAME', frame)
    cv2.setMouseCallback('FRAME', draw)
    Key = cv2.waitKey(100) & 0xFF
    if Key == ord('s'):
        with open("freedomtech", "wb") as f:

```

```

        data = {'polylines': polylines, 'area_names':
area_names}

        pickle.dump(data, f)

```

### 7.3.5 Section 5: Displaying the Live Camera Stream

In this final section, another loop is initiated to display the live camera stream. Frames are continuously read from the camera using `cap.read()`. If a frame is successfully obtained, it is displayed in a window titled "Camera Stream" using `cv2.imshow()`. The loop waits for the 'q' key to be pressed (identified by `ord('q')`) to exit the loop. Once the loop ends, the `VideoCapture` object is released using `cap.release()`, and all windows are closed using `cv2.destroyAllWindows()`. This section enables the display of the live camera stream and terminates when the user presses the 'q' key.

#### Python Code section -

```

while True:
    ret, frame = cap.read()
    if not ret:
        break
    cv2.imshow("Camera Stream", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()

```

### 7.3.6 Result

The obtained results reveal the identified regions for both outgoing and incoming traffic. These marked areas are vital for the image recognition process, as they serve as reference points for accurately detecting and tracking vehicles as they cross these designated zones. By leveraging this information, the system can effectively analyze the traffic flow and provide valuable insights for improved traffic management strategies.





Figure 7-1 - Demonstrating Marked Points in the Highway

## 7.4 Firmware Development – Traffic Count Detection

We employ image recognition technology at designated points along the traffic route to accurately count the number of vehicles passing through specific areas. These marked areas serve as crucial checkpoints for capturing visual data. Once the image recognition is performed, the relevant information is seamlessly transmitted to a cloud-based MySQL database for secure storage and further analysis. Below, we concisely explain the underlying code used for this purpose.

### 7.4.1 Section 1: Imports and Data Loading

The code begins by importing necessary libraries such as cv2 (OpenCV), numpy, pickle, pandas, ultralytics, YOLO, cvzone, requests, datetime, and time. These libraries provide various functionalities for image processing, data manipulation, object detection, and web communication. The script then loads some pre-existing data from files, including freedomtech and coco.txt, using the pickle and open functions.

#### Python Code section -

```
import cv2
import numpy as np
import pickle
import pandas as pd
from ultralytics import YOLO
```

```

import cvzone
import requests
import datetime
import time

# Load pre-existing data
with open('freedomtech', 'rb') as f:
    freedomtech = pickle.load(f)

with open('coco.txt', 'r') as f:
    coco_classes = f.read().split('\n')

```

### 7.4.2 Section 2: Model Initialization

The code initializes the YOLO model by creating an instance of YOLO from the ultralytics module. The model is loaded with the weights stored in the file yolov8s.pt.

#### Python Code section -

```

# Initialize YOLO model
model = YOLO("yolov8s.pt")

```

### 7.4.3 Section 3: Video Capture Configuration

The code sets up video capture using `cv2.VideoCapture(1)`, which captures frames from a camera with index 1. The script also sets the width and height of the captured frames to 640 and 480 pixels, respectively, using the `cap.set` function.

#### Python Code section -

```

# Configure video capture
cap = cv2.VideoCapture(1)
cap.set(3, 640) # Set width
cap.set(4, 480) # Set height

```

#### 7.4.4 Section 4: Frame Processing and Object Detection

Inside the main loop, the code continuously captures frames from the video feed. It resizes each frame to a width of 1020 pixels and a height of 500 pixels using `cv2.resize`. The resized frame is stored in `frame_copy` for further processing. The YOLO model then predicts objects in the frame using `model.predict(frame)`, which returns the detected objects' bounding boxes.

##### Python Code section -

```
while True:
    # Capture frame
    ret, frame = cap.read()

    # Resize frame
    frame_copy = cv2.resize(frame, (1020, 500))

    # Object detection
    results = model.predict(frame_copy)
    boxes = results.xyxy[0][:, :4].numpy()
    class_labels = results.xyxy[0][:,
5].numpy().astype(int)
```

#### 7.4.5 Section 5: Object Classification and Position Analysis

The code extracts the bounding box coordinates and corresponding class labels from the YOLO model's predictions. It iterates over each detected object and checks its class label. If the object is classified as a car, truck, or bus, the code determines whether it is an incoming or outgoing vehicle based on its x-coordinate. The coordinates and classification information are stored in separate lists accordingly. Additionally, rectangles are drawn around the detected vehicles on the frame using `cv2.rectangle`.

##### Python Code section -

```
# Object classification and position analysis
//store vehicle count = []
for i, box in enumerate(boxes):
    class_label = coco_classes[class_labels[i]]
    x, y, w, h = box
```

```

# Draw rectangle around detected object
cv2.rectangle(frame_copy, (x, y), (w, h), (255,
0, 0), 2)

# Vehicle classification and position analysis
if class_label in ['car', 'truck', 'bus']:
    if x < freedomtech["line1"]:
        if class_label == 'car':
            outgoing_cars.append((x, y, w, h))
        elif class_label == 'truck':
            outgoing_trucks.append((x, y, w, h))
        elif class_label == 'bus':
            outgoing_buses.append((x, y, w, h))
    elif x > freedomtech["line2"]:
        if class_label == 'car':
            incoming_cars.append((x, y, w, h))
        elif class_label == 'truck':
            incoming_trucks.append((x, y, w, h))
        elif class_label == 'bus':
            incoming_buses.append((x, y, w, h))

```

#### 7.4.6 Section 6: Vehicle Counting and Visualization

This section counts the vehicles based on their positions relative to predefined polylines, which represent areas of interest. The code iterates over each polyline and checks if any detected vehicle's center point lies within the polyline using `cv2.pointPolygonTest`. If a vehicle is found within a polyline, a circle is drawn around it, and the polyline is highlighted on the frame. The counts of outgoing and incoming cars, trucks, and buses are stored in separate lists. The total counts and free space (areas without vehicles) are calculated.

##### Python Code section –

```

# Vehicle counting and visualization

outgoing_count = len(outgoing_cars) +
len(outgoing_trucks) + len(outgoing_buses)

```

```

    incoming_count = len(incoming_cars) +
len(incoming_trucks) + len(incoming_buses)

    fullcount = outgoing_count + incoming_count

    # Calculate free space
    free_space = freedomtech["area"] - fullcount

    # Iterate over polylines
    for poly in freedomtech["polyline"]:
        points = np.array(poly, np.int32)
        points = points.reshape((-1, 1, 2))

        # Check if any vehicle's center point lies within
the polyline
        for vehicle in outgoing_cars + outgoing_trucks +
outgoing_buses + incoming_cars + incoming_trucks +
incoming_buses:
            x, y, w, h = vehicle
            center_x = (x + w) // 2
            center_y = (y + h) // 2
            if cv2.pointPolygonTest(points, (center_x,
center_y), False) == 1:
                # Draw circle around vehicle
                cv2.circle(frame_copy, (center_x,
center_y), 5, (0, 255, 0), -1)

                # Highlight polyline
                cv2.polylines(frame_copy, [points], True,
(0, 255, 0), 2)

```

### 7.4.7 Section 7: Displaying Counters

The code uses `cvzone.putTextRect` to display the vehicle counters and related information on the frame. This information includes the counts of outgoing and incoming cars, trucks, and buses. The text is positioned at specific coordinates on the frame using `cvzone.putTextRect`.

### Python Code section -

```
cvzone.putTextRect(frame_copy, f"Outgoing Cars:
{len(outgoing_cars)}", (20, 20), 1, 1, 2)

cvzone.putTextRect(frame_copy, f"Incoming Cars:
{len(incoming_cars)}", (20, 60), 1, 1, 2)

cvzone.putTextRect(frame_copy, f"Outgoing Trucks:
{len(outgoing_trucks)}", (20, 100),Section 7: Displaying
Counters

```python

cvzone.putTextRect(frame_copy, f"Incoming Trucks:
{len(incoming_trucks)}", (20, 140), 1, 1, 2)

cvzone.putTextRect(frame_copy, f"Outgoing Buses:
{len(outgoing_buses)}", (20, 180), 1, 1, 2)

cvzone.putTextRect(frame_copy, f"Incoming Buses:
{len(incoming_buses)}", (20, 220), 1, 1, 2)

cvzone.putTextRect(frame_copy, f"Free Space:
{free_space}", (20, 260), 1, 1, 2)
```

### 7.4.8 Section 8: Data Transmission

If there is at least one vehicle detected (i.e., `fullcount > 0`), the code sends the vehicle count data to a PHP script located at [https://porsgruntraffic.com/Insert\\_data.php](https://porsgruntraffic.com/Insert_data.php) using a POST request. The data is sent in JSON format, including counts for outgoing and incoming cars, trucks, buses, and free space. The response from the server is checked, and a success or error message is printed. A slight delay of 0.2 seconds is introduced using `time.sleep`. Finally, the processed frame is displayed using `cv2.imshow`, and the loop continues until the 'q' key is pressed.

The script releases the video capture resources and closes the windows when the loop is terminated, ensuring proper cleanup.

Please note that without the `freedomtech` and `coco.txt` files and access to the specified PHP script, the code's complete functionality cannot be demonstrated.

### Python Code section -

```
# Data transmission

if fullcount > 0:

    data = {

        'outgoing_cars': len(outgoing_cars),
```

```

        'incoming_cars': len(incoming_cars),
        'outgoing_trucks': len(outgoing_trucks),
        'incoming_trucks': len(incoming_trucks),
        'outgoing_buses': len(outgoing_buses),
        'incoming_buses': len(incoming_buses),
        'free_space': free_space,
        'place': 'Hovenga'
    }

    response =
requests.post('https://porsgrunntraffic.com/Insert_data.p
hp', json=data

        if response.status_code == 200:
            print("Data sent successfully!")
        else:
            print("Error sending data.")
        time.sleep(0.2)

# Display processed frame
cv2.imshow("Vehicle Detection", frame_copy)
# Exit loop if 'q' key is pressed
if cv2.waitKey(1) == ord('q'):
    break

# Release video capture resources and close windows
cap.release()
cv2.destroyAllWindows()

```

### 7.4.9 Result

The system accurately counts incoming and outgoing vehicles using image processing. Real-time data is displayed on the edge computer, while the information is seamlessly sent to a Cloud MySQL database for secure storage and analysis, enabling informed decision-making and optimized traffic management. The below Figure 7-2 shows the interface of the

monitored system.

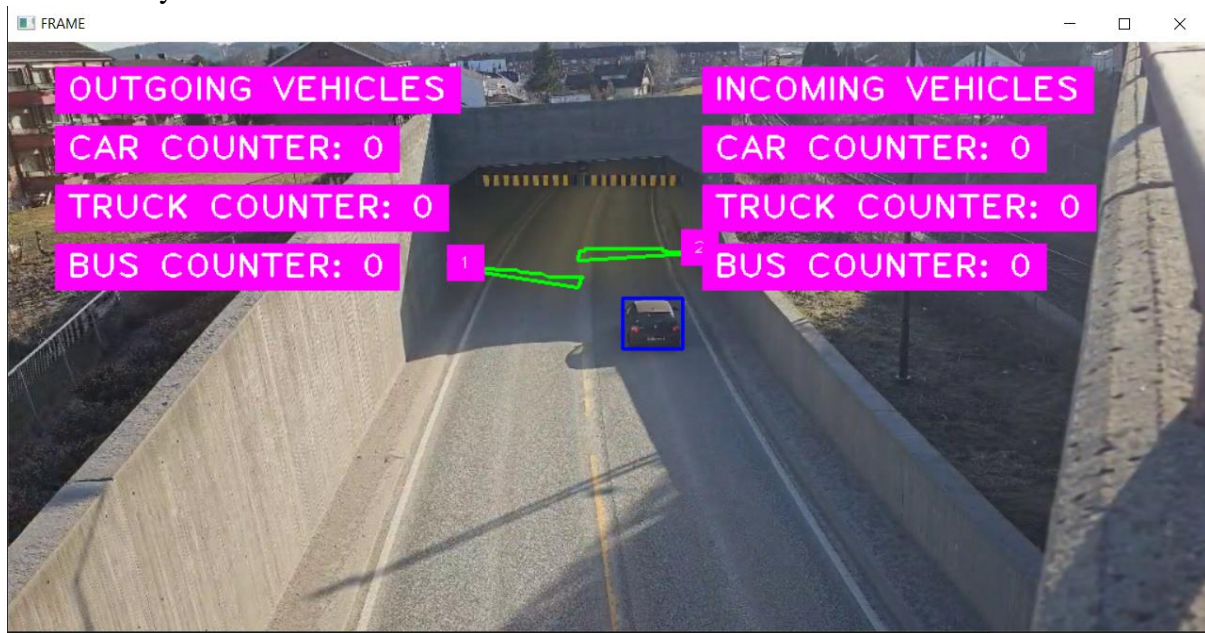


Figure 7-2 - Monitor Interface with Counters

## 7.5 Server-side Code Implementation

For this server-side scripting a PHP programming platform is utilized. PHP is a versatile server-side scripting language widely used in web development and database handling. It offers seamless integration with HTML, enabling dynamic content generation and interaction. With its extensive libraries and frameworks, PHP provides a robust environment for building web applications, handling form submissions, and accessing databases. Its simplicity, flexibility, and cross-platform compatibility make PHP a popular choice for developers. PHP's ability to handle various databases, including MySQL, PostgreSQL, and Oracle, makes it suitable for managing data-driven applications. Overall, PHP empowers developers to create dynamic and interactive web experiences efficiently. Below are the code for data retrieval and storing

### 7.5.1 Data Retrieval and Variable Assignment

In the first paragraph, the code retrieves the vehicle counts and other data from the POST request. The values are accessed using the `$_POST` super global and assigned to corresponding variables. For example, `$carCountOutgoing` stores the value of 'car\_count\_outgoing' sent via the POST request. Similarly, the other counts and data are retrieved and assigned to their respective variables.

```
$carCountOutgoing = $_POST['car_count_outgoing'];  
$truckCountOutgoing = $_POST['truck_count_outgoing'];  
$busCountOutgoing = $_POST['bus_count_outgoing'];
```



```

$freeSpaceOutgoing = $_POST['free_space_outgoing'];
$carCountIncoming = $_POST['car_count_incoming'];
$truckCountIncoming = $_POST['truck_count_incoming'];
$busCountIncoming = $_POST['bus_count_incoming'];
$freeSpaceIncoming = $_POST['free_space_incoming'];

```

## 7.5.2 Database Connection and Error Handling

In the second paragraph, the code establishes a connection to the database using PDO (PHP Data Objects). It creates a new PDO instance by providing the necessary connection parameters: host, database name, username, and password. Error handling is implemented using a try-catch block, where any exceptions thrown during the connection attempt are caught and an error message is displayed.

```

$host = 'localhost';
$dbName = 'u607713813_traffic';
$username = 'u607713813_porsgrunn';
$password = '*****';
try {
    $pdo = new PDO("mysql:host=$host;dbname=$dbName",
$username, $password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die("Error: " . $e->getMessage());
}

```

## 7.5.3 SQL Statement Preparation, Execution, and Error Handling

In the final paragraph, the code prepares an SQL statement to insert the retrieved data into the database. The SQL statement includes placeholders denoted by colons (e.g., :timestamp, :carCountOutgoing) for the values that will be inserted. The code then uses the bindParam method to bind the values from the variables to the corresponding placeholders in the SQL statement. After that, the SQL statement is executed using the execute method. If the execution is successful, a success message “Data inserted successfully” is echoed. If any errors occur during the execution, they are caught, and an error message is displayed.

## PHP code -

```
$sql = "INSERT INTO vehicle_counts (timestamp,
car_count_outgoing, truck_count_outgoing,
bus_count_outgoing, free_space_outgoing,
car_count_incoming, truck_count_incoming,
bus_count_incoming, free_space_incoming,place) VALUES
(:timestamp, :carCountOutgoing, :truckCountOutgoing,
:busCountOutgoing, :freeSpaceOutgoing, :carCountIncoming,
:truckCountIncoming, :busCountIncoming,
:freeSpaceIncoming, :place)";

// Bind the values to the parameters in the SQL statement
$stmt = $pdo->prepare($sql);
$stmt->bindParam(':timestamp', $timestamp);
$stmt->bindParam(':load data stored');

// Execute the SQL statement
try {
    $stmt->execute();
    echo "Data inserted successfully.";
} catch (PDOException $e) {
    die("Error: " . $e->getMessage());
}
?>
```

## 7.6 Developing the Tableau Dashboard

The below Figure 7-3 demonstrated the result of the Traffic management system. It has the capability of drilling down the data as the filters are set up to it. These dashboards are useful for decision making purposes.

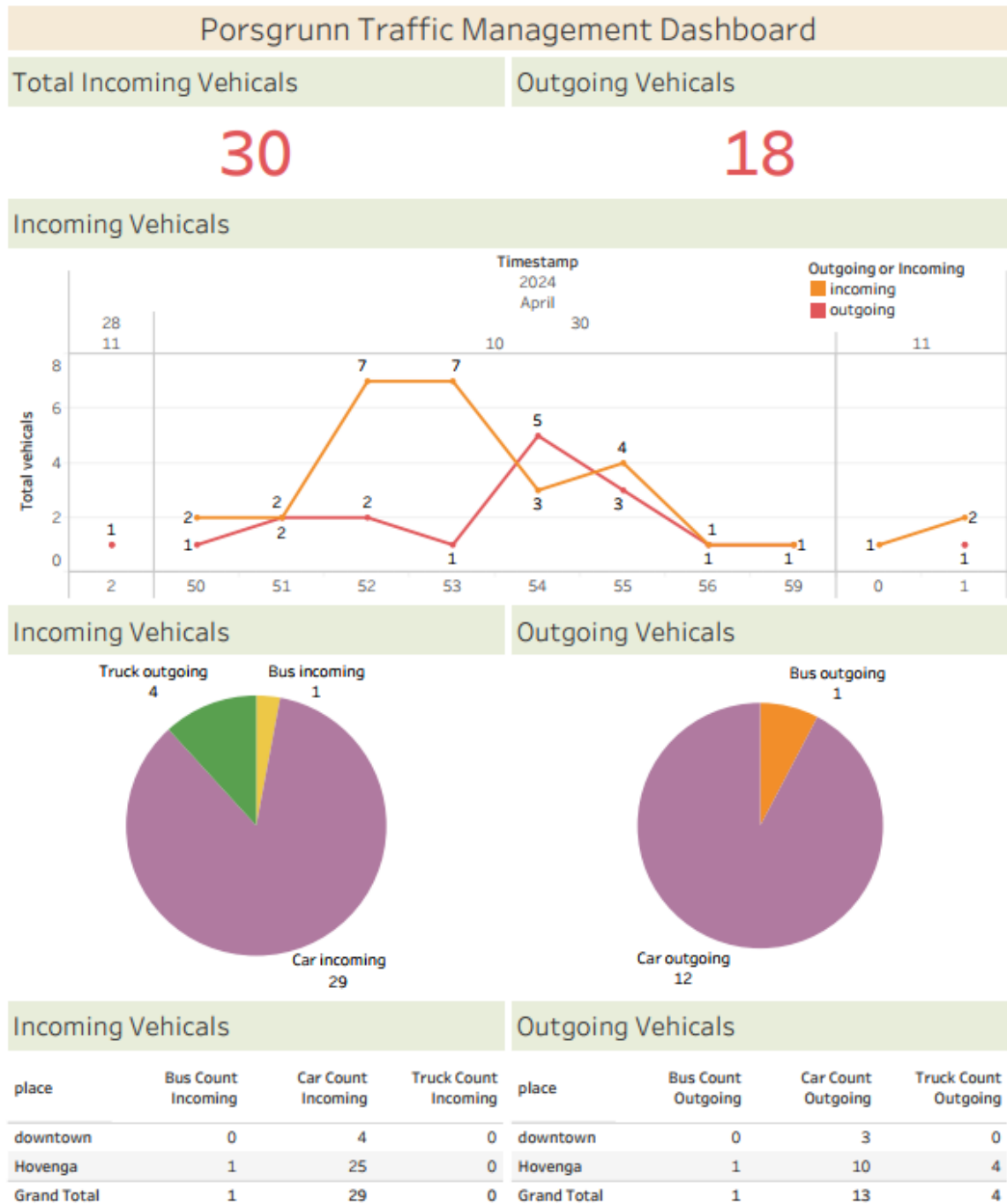


Figure 7-3 - Developed Porsgrunn Traffic Management Dashboard

# 8 Implementation Smart Parking System

## 8.1 Introduction

The development of smart cities has gained significant momentum in recent years, aiming to enhance urban functionality, sustainability, and quality of life for its residents. One critical aspect of urban living is efficient parking management, which often poses challenges such as congestion, wasted time, and increased pollution. To address these issues, this project proposes the implementation of a smart parking system for Porsgrunn Kommune, leveraging the power of Python image recognition technology. This project's main objective is to create an intelligent parking system that uses advanced image recognition techniques to enable efficient parking space detection and management. By leveraging Python programming language and its vast array of libraries and frameworks, we can develop a robust and scalable system that can accurately identify available parking spaces in real time.

The system will employ cameras strategically placed in parking lots throughout Porsgrunn kommune to capture live images. These images will then be processed using image recognition algorithms to detect and analyze parking spaces' occupancy status. By applying computer vision techniques, the system can differentiate between occupied and vacant parking spots, providing accurate and up-to-date information to drivers in real time. The Python image recognition system will offer numerous benefits to both drivers and Porsgrunn kommune. Drivers will have access to a mobile or web application that displays the availability of parking spaces in real-time, enabling them to locate and secure a parking spot quickly. This will reduce the time spent searching for parking, alleviate traffic congestion, and enhance the overall parking experience.

For Porsgrunn kommune, the smart parking system will provide valuable data insights, such as parking occupancy rates, peak usage times, and parking space utilization trends. This information will enable the municipality to optimize parking infrastructure planning, improve traffic flow, and make data-driven decisions regarding parking policies and pricing structures. In conclusion, the proposed smart parking system utilizing Python image recognition technology presents an innovative solution to address the parking challenges faced by Porsgrunn kommune. By leveraging the capabilities of Python and image recognition algorithms, this project aims to optimize parking space utilization, enhance the parking experience for drivers, and contribute to the overall development of a smarter and more efficient city.

## Marking parking points

To enhance the efficiency of vehicle parking management, we have developed a system that allows for the identification and marking of custom locations for parking slots using image recognition. This innovative approach enables accurate monitoring and optimization of parking spaces.

The process is as follows sections.

### 8.1.1 Section 1: Importing Libraries and Setting Up

This section imports the necessary libraries for the code: `cv2` for computer vision tasks, `numpy` for numerical operations, `cvzone` for additional drawing functionalities, and `pickle` for object serialization. The code then initializes variables including `cap` which is a `VideoCapture` object used to read frames from the camera (in this case, the default camera index 0). It also sets the initial values of `drawing` and `area\_names`.

#### Python Code section -

```
import cv2

import numpy as np

import cvzone

import pickle

cap = cv2.VideoCapture(0)

drawing = False

area_names = []
```

### 8.1.2 Section 2: Loading Pre-defined Polygons and Area Names

In this section, the code attempts to load pre-defined polygons and area names from a pickled file named "freedomtech". It uses a `try-except` block to handle the case where the file is not found or loading fails. If the loading is successful, the values are assigned to the `polygons` and `area\_names` variables. If loading fails or the file is not found, `polygons` is initialized as an empty list.

#### Python Code section -

```
try:

    with open("freedomtech", "rb") as f:
```

```

        data = pickle.load(f)
        polylines, area_names = data['polylines'],
data['area_names']
except:
    polylines = []

```

### 8.1.3 Section 3: Mouse Event Handler for Drawing Polyines

This section defines the `draw` function, which serves as the mouse event handler. It collects the points for drawing polyines when the left mouse button is clicked (`LBUTTONDOWN`). When the mouse is moved (`MOUSEMOVE`), it appends the current point to the `points` list if drawing is in progress. When the left mouse button is released (`LBUTTONUP`), it sets `drawing` to `False`, prompts the user to enter an area name using the `input` function, and appends the area name to `area_names` if a non-empty name is provided. Finally, it converts the collected points into a NumPy array and appends it to the `polylines` list.

#### Python Code section -

```

points = []
current_name = ""

def draw(event, x, y, flags, param):
    global points, drawing

    drawing = True

    if event == cv2.EVENT_LBUTTONDOWN:
        points = [(x, y)]
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing:
            points.append((x, y))
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        current_name = input('areaname:-')
        if current_name:

```

```

        area_names.append(current_name)
    polylines.append(np.array(points, np.int32))

```

### 8.1.4 Section 4: Drawing Polylines and Displaying Area Names

In this section, the code enters a loop that continuously reads frames from the camera. It resizes the frame to a width of 1020 and a height of 500 pixels. Then, it iterates over the pre-defined `polylines` and `area\_names`. For each polyline, it draws the polyline on the frame using `cv2.polylines` and displays the corresponding area name using `cvzone.putTextRect`. The modified frame is displayed in a window titled 'FRAME' using `cv2.imshow`. The mouse callback function is set to `draw` using `cv2.setMouseCallback` to enable drawing new polylines and collecting area names. The loop waits for a key press (100ms delay) and if the 's' key is pressed, it saves the `polylines` and `area\_names` to the "freedomtech" file using pickle.

#### Python Code section -

```

while True:
    ret, frame = cap.read()
    if not ret:
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
        continue
    frame = cv2.resize(frame, (1020, 500))
    for i, polyline in enumerate(polylines):
        cv2.polylines(frame, [polyline], True, (0, 0,
255), 2)
        cvzone.putTextRect(frame, f'{area_names[i]}',
tuple(polyline[0]), 1, 1)
    cv2.imshow('FRAME', frame)
    cv2.setMouseCallback('FRAME', draw)
    Key = cv2.waitKey(100) & 0xFF
    if Key == ord('s'):
        with open("freedomtech", "wb") as f:
            data = {'polylines': polylines, 'area_names':
area_names}
            pickle.dump(data, f)

```

### 8.1.5 Section 5: Displaying the Live Camera Stream

In this final section, the code enters another loop to display the live camera stream. It continuously reads frames from the camera using `cap.read()`. If a frame is successfully read, it displays the frame in a window titled "Camera Stream" using `cv2.imshow()`. The loop waits for the 'q' key to be pressed (`ord('q')`) to exit the loop. Once the loop ends, the `VideoCapture` object is released using `cap.release()`, and all windows are closed with `cv2.destroyAllWindows()`. This section allows for displaying the live camera stream and terminates when the user presses the 'q' key.

#### Python Code section -

```
while True:
    ret, frame = cap.read()
    if not ret:
        break
    cv2.imshow("Camera Stream", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

### 8.1.6 Results

The below results in Figure 8-1 are to demonstrate the marked areas of the locations.

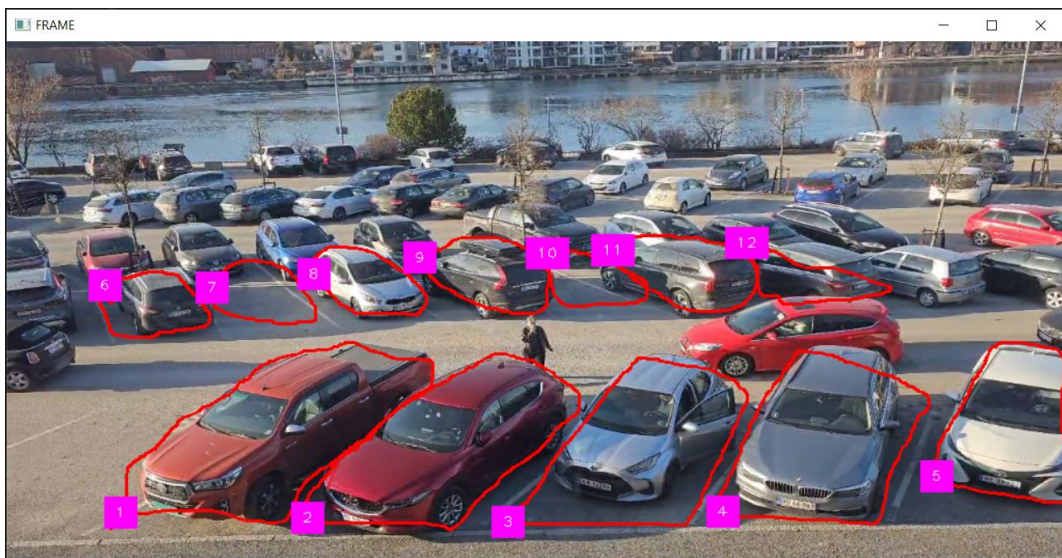


Figure 8-1 - Downtown Parking area Parking Slots



## 8.2 Firmware Development – Vehicle Parking Detection

The code uses YOLO object detection to count cars and monitor parking spaces. It reads pre-defined polylines and area names from a pickled file. Frames from a video file are processed, and cars are detected using YOLO. The script checks if cars intersect with the pre-defined polygons and updates the car count and free space count. Changes are sent to a PHP file with the current time. The counts are displayed on the frame, and the processed frames are shown.

### 8.2.1 Section 1: Import Libraries and Load Data

#### Python code

```
import cv2
import numpy as np
import pickle
import pandas as pd
from ultralytics import YOLO
import cvzone
import requests
import datetime

with open("freedomtech", "rb") as f:
    data = pickle.load(f)
    polylines, area_names = data['polylines'],
    data['area_names']

my_file = open("coco.txt", "r")
data = my_file.read()
class_list = data.split("\n")
```

This section imports the required libraries and loads data from files. It imports OpenCV (cv2), NumPy (numpy), pickle (pickle), pandas (pd), YOLO from Ultralytics (YOLO), cvzone, requests, and datetime. It then opens the "freedomtech" file, loads the data into variables polylines and area\_names, and reads the contents of the "coco.txt" file into the class\_list variable.

### 8.2.2 Section 2: Initialize Variables and Model

#### Python code

```
model = YOLO('yolov8s.pt')

cap = cv2.VideoCapture('Parking.mp4')

count = 0
prev_car_count = 0
```

```
prev_free_space = 0
```

This section initializes variables such as model with the YOLO model using 'yolov8s.pt' weights, cap with a VideoCapture object for 'Parking.mp4', and count, prev\_car\_count, and prev\_free\_space with initial values.

### 8.2.3 Section 3: Process Video Frames

#### Python code

```
while True:
    ret, frame = cap.read()
    if not ret:
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
        continue

    count += 1
    if count % 3 != 0:
        continue
```

This section starts a loop that processes the video frames. It reads a frame from the cap object, checks if the frame was successfully read, and skips to the next iteration if not. It increments the count variable by 1 and continues to the next iteration if the count is not divisible by 3.

### 8.2.4 Section 4: Object Detection and Car Extraction

#### Python code

```
frame = cv2.resize(frame, (1020, 500))
frame_copy = frame.copy()
results = model.predict(frame)
a = results[0].boxes.data
px = pd.DataFrame(a).astype("float")
list1 = []
for index, row in px.iterrows():
    x1 = int(row[0])
    y1 = int(row[1])
    x2 = int(row[2])
    y2 = int(row[3])
    d = int(row[5])

    c = class_list[d]
    cx = int(x1 + x2) // 2
    cy = int(y1 + y2) // 2
    if 'car' in c:
        list1.append([cx, cy])
```

In this section, the frame is resized, and a copy of the frame is created. The YOLO model is used to predict objects in the frame, and the bounding box information is extracted. The bounding boxes are filtered to extract cars, and their centroids are stored in list1.

## 8.2.5 Section 5: Process Detected Cars and Parking Areas

### Python code

```
counter1 = []
list2 = []
for I, polyline in enumerate(polylines):
    list2.append(i)
    cv2.polylines(frame, [polyline], True, (0, 255, 0), 2)
    cvzone.putTextRect(frame, f'{area_names[i]}',
tuple(polyline[0]), 1, 1)
    for i1 in list1:
        cx1 = i1[0]
        cy1 = i1[1]
        result = cv2.pointPolygonTest(polyline, ((cx1,
cy1)), False)
        if result >= 0:
            cv2.circle(frame, (cx1, cy1), 5, (255, 0, 0),
-1)
            cv2.polylines(frame, [polyline], True, (0, 0,
255), 2)
            counter1.append(cx1)
```

This section iterates over the parking areas defined by polylines and their corresponding indices. It draws the parking areas on the frame and adds the area names as text. For each car centroid in list1, it checks if the centroid is within a parking area using cv2.pointPolygonTest. If the centroid is inside a parking area, it highlights the area and adds the centroid's x-coordinate to counter1.

## 8.2.6 Section 6: Calculate Car Count and Free Space

### Python code

```
car_count = len(counter1)

free_space = len(list2) - car_count
```

The car count is calculated by taking the length of counter1, and the free space is calculated by subtracting the car count from the total number of parking areas (list2).

## 8.2.7 Section 7: Send Data to PHP File and Update Previous Counts

### Python code

```
    if car_count != prev_car_count or free_space !=
prev_free_space:
        # Get current time
        current_time =
datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        # Send values to PHP file
        data = {'car_count': car_count, 'free_space':
free_space, 'time': current_time, 'place': 'downtown'}
        response =
requests.post('https://porsgrunntraffic.com/Insert_parkin
g_data.php', data=data)
        print(response.text) # Print the response from
the PHP file

        prev_car_count = car_count
        prev_free_space = free_space
```

### Send Data to PHP File and Update Previous Counts

If the car count or free space has changed since the last iteration, the current time is obtained using `datetime.datetime.now()`. The values are then sent to a PHP file ([https://porsgrunntraffic.com/Insert\\_parking\\_data.php](https://porsgrunntraffic.com/Insert_parking_data.php)) via a POST request using the `requests` library. The response from the PHP file is printed, and the previous car count and free space are updated with the current values.

## 8.2.8 Section 8: Display Results and Handle Key Events

### Python code

```
    cvzone.putTextRect(frame, f'CARCOUNTER:-{car_count}',
(50, 50), 2, 2)
    cvzone.putTextRect(frame, f'Free_SPACE:-
{free_space}', (50, 160), 2, 2)
    cv2.imshow('FRAME', frame)
    key = cv2.waitKey(1) & 0xFF

    cap.release()
    cv2.destroyAllWindows()
```

In this section, the car count and free space are displayed on the frame using `cvzone.putTextRect`. The frame is displayed in a window titled 'FRAME' using `cv2.imshow`. The code waits for a key event and stores the key pressed. The loop continues until interrupted. Finally, the code releases the video capture (`cap`) and closes all OpenCV windows (`cv2.destroyAllWindows()`).

## 8.2.9 Results

The obtained results showcase the car count and the availability of free space at the location. The designated areas are marked in red, indicating occupied spaces, while green represents areas with available parking. The recognized frames are highlighted in blue, with the frame turning red when it enters the marked area. This visual representation aids in monitoring parking occupancy and facilitating efficient management.



Figure 8-2 - Display the Monitor with Counters

## 8.3 Server-side Code Implementation

For this project an utilized the PHP programming platform for server-side scripting is used. PHP is a highly versatile language commonly used in web development and database management. It seamlessly integrates with HTML, enabling the generation of dynamic content and facilitating user interaction. With its comprehensive range of libraries and frameworks, PHP provides a robust environment for creating web applications, processing form submissions, and accessing databases. Its simplicity, flexibility, and compatibility across different platforms have contributed to its widespread adoption among developers. PHP's capability to handle various databases, such as MySQL, PostgreSQL, and Oracle, makes it well-suited for developing data-driven applications. In summary, PHP empowers developers to efficiently build dynamic and interactive web experiences. Below, you will find the code snippets for data retrieval and storage.

### 8.3.1 Handling Form Data and Database Connection

In this section, the code receives data from a form using the `$_POST` superglobal. It assigns the values of `car_count`, `free_space`, `time`, and `place` to corresponding variables. The code then establishes a database connection using the provided server name, username, password, and database name. It creates a new instance of the `mysqli` class and checks if the connection was successful. If the connection fails, it terminates the script and displays an error message.

```
<?php
```

```
$car_count = $_POST['car_count'];
$free_space = $_POST['free_space'];
$time = $_POST['time'];
$place = $_POST['place'];

$servername = 'localhost';
$username = 'u607713813_porsgrunnpark';
$password = '*****';
$dbname = 'u607713813_parking';

$conn = new mysqli($servername, $username, $password,
$dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

### 8.3.2 Section 2: Inserting Data into the Database

In this section, the code constructs an SQL INSERT statement using the received form data. The SQL statement is stored in the \$sql variable and specifies the table name (parking\_data) and column names (car\_count, free\_space, time, place) where the data should be inserted. The values from the form variables are inserted into the SQL statement using variable interpolation. The code then executes the SQL query using the query() method of the database connection object. If the query execution is successful, it displays the message "Data inserted successfully". Otherwise, it displays an error message along with the specific SQL query and the error information.

```
$sql = "INSERT INTO parking_data (car_count, free_space,
time, place) VALUES ('$car_count', '$free_space',
'$time', '$place')";

if ($conn->query($sql) === TRUE) {
    echo "Data inserted successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
```

### 8.3.3 Section 3: Closing the Database Connection

In this final section, the code closes the database connection using the close() method of the database connection object. This ensures that the connection is properly terminated after executing the SQL query and handling any potential errors.

```
$conn->close();?>
```

## 8.4 Developing the Tableau Dashboard

Figure 8-3 illustrates the outcome of the Smart parking system, showcasing its ability to drill down into data as filters are applied. These interactive dashboards serve as valuable tools for making informed decisions.

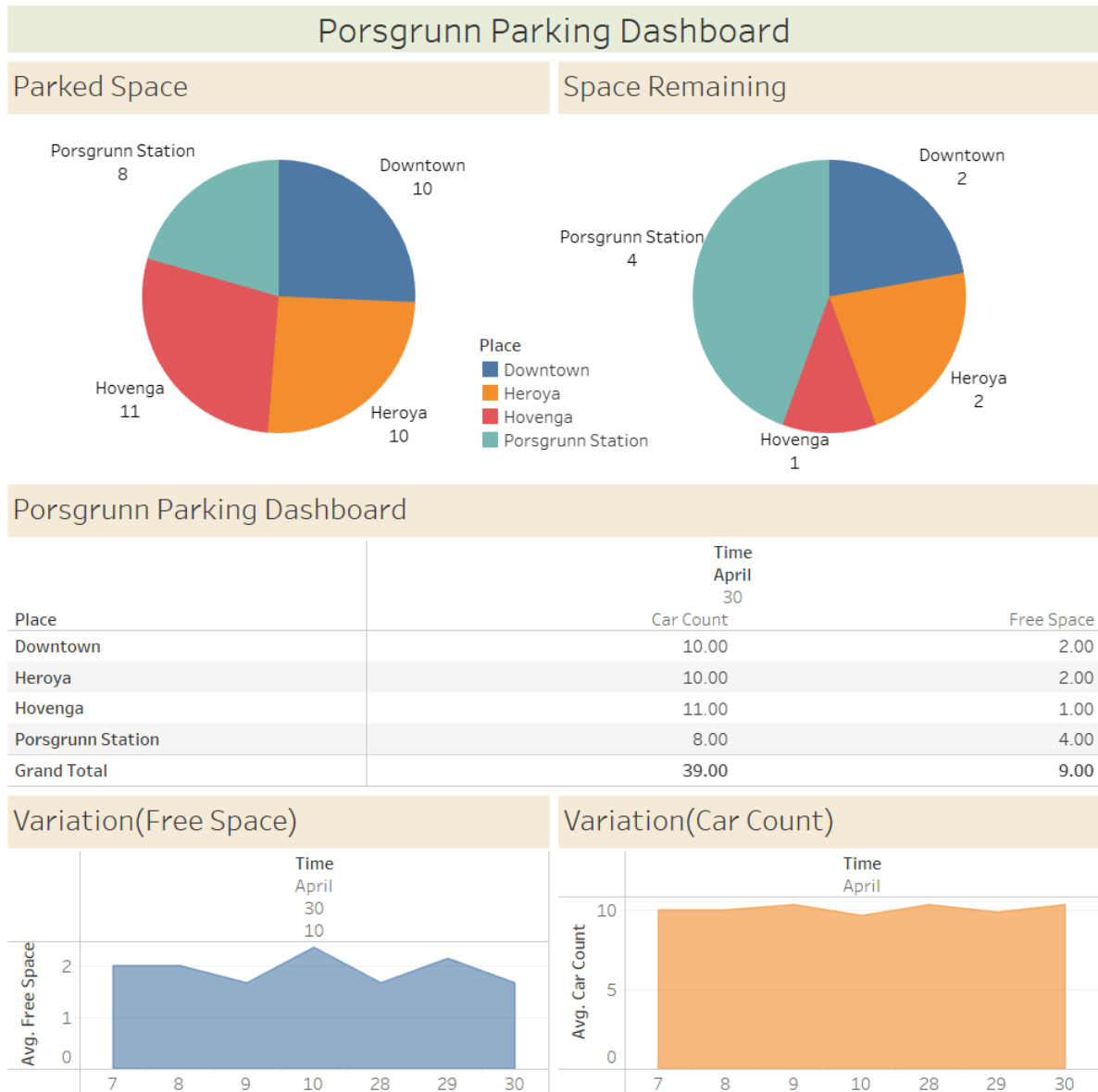


Figure 8-3 - Porsgrunn Parking Dashboard

# 9 Providing Real-Time Updates for the Community

A real-time data updating website is crucial for the image processing IoT traffic management system as it ensures that users have access to the latest information on traffic conditions, parking availability, and system functionality, enabling effective decision-making and enhancing overall traffic management efficiency. The below Figure 9-1 demonstrates a prototype to the end users who are interested in analyzing the data.

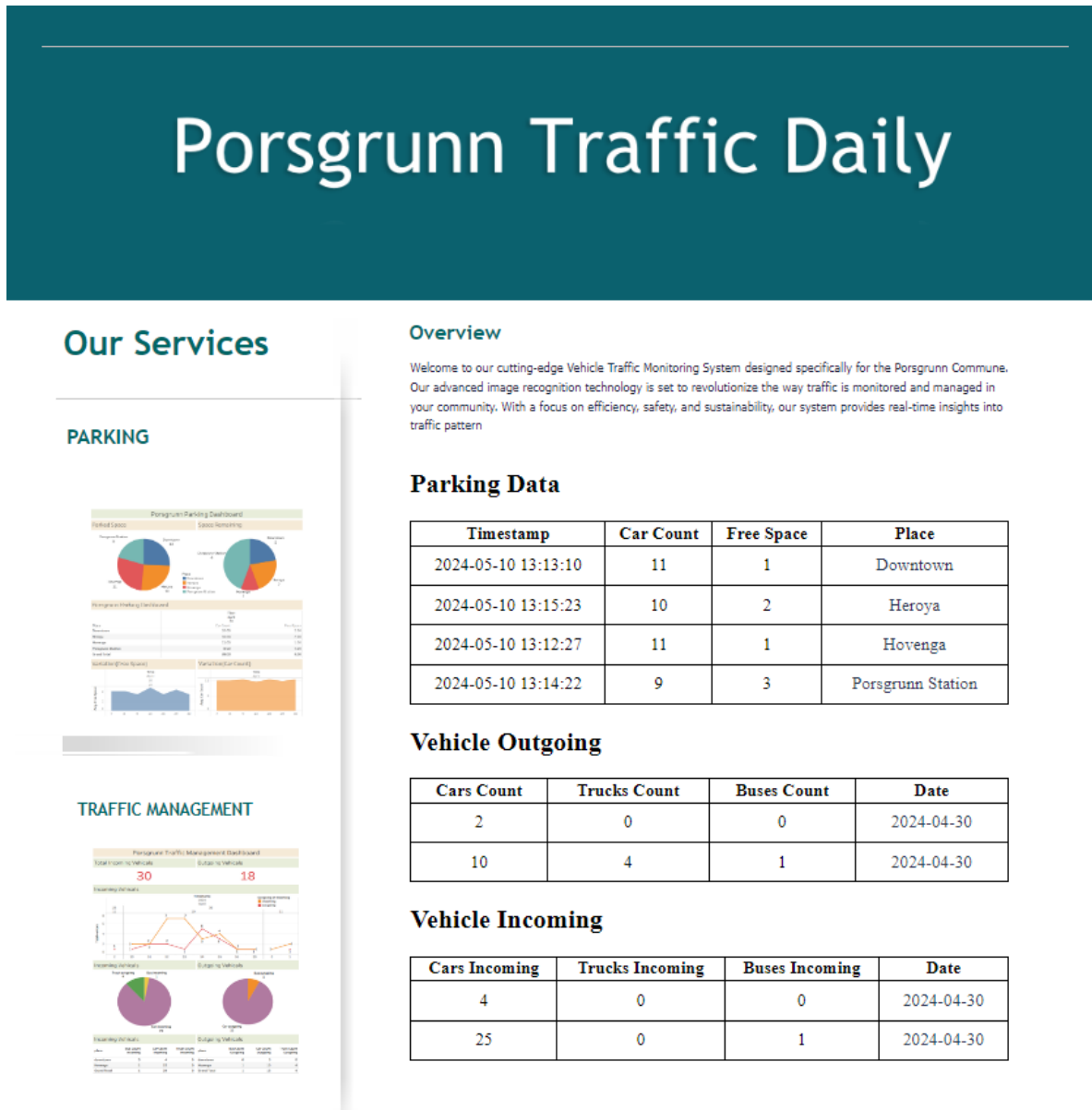


Figure 9-1 - Realtime Data Updating Website.



## 9.1 Traffic Data for The Community

This traffic counting feature offers significant advantages for traffic management and urban planning. City authorities can gather real-time data on the volume and composition of vehicles entering the city, allowing them to understand traffic patterns, identify congestion hotspots, and optimize traffic flow accordingly. This information is invaluable for implementing effective transportation policies, improving road infrastructure, and enhancing overall traffic management strategies.

The use of image recognition technology eliminates the need for manual counting, making the process more efficient and accurate. It also enables historical data collection, allowing for long-term analysis and trend identification. By harnessing this technology, cities can make data-driven decisions to alleviate congestion, reduce travel times, and enhance the overall transportation experience for residents and visitors. The below Figure 9-2 - Vehicle Counting section of the shows how it has been represented.

### Vehicle Outgoing

Cars Count	Trucks Count	Buses Count	Date
2	0	0	2024-04-30
10	4	1	2024-04-30

### Vehicle Incoming

Cars Incoming	Trucks Incoming	Buses Incoming	Date
4	0	0	2024-04-30
25	0	1	2024-04-30

Figure 9-2 - Vehicle Counting section of the Website.

Once the date is selected, the data will redirect it to an area's breakdown page, which will give a real-time detail of the traffic as Figure 9-3.

Date: 2024-04-28
[Back](#)

---

Below is a representation of the outgoing traffic in the past few days. The tabular format provides a clear overview of the outgoing traffic on each respective day, enabling easy comparison and analysis of the traffic patterns.

### Traffic Status

Date	Place	Total Car Count Outgoing	Total Truck Count Outgoing	Total Bus Count Outgoing
2024-04-28	downtown	1	0	0

Figure 9-3 - Area Breakdown Page of Traffic Management System

## 9.2 Smart Parking System

The smart parking system described here offers real-time information about available and occupied parking slots. It is seamlessly integrated with a dedicated website, allowing users to access data on parking availability in Porsgrunn city. By knowing which areas have free parking, users can plan their routes, accordingly, resulting in reduced traffic congestion and more efficient parking for vehicles.

This system utilizes image recognition technology to capture and analyze parking slot occupancy. This process accurately determines whether a slot is occupied or vacant. The system then updates a central database with this information, ensuring that the parking data displayed on the website remains current.

This smart parking system contributes to a more streamlined and convenient parking experience by providing users with real-time parking information. It helps drivers quickly locate available parking spaces, reducing the time spent searching for a spot and minimizing traffic disruptions caused by vehicles circling in search of parking. Ultimately, this system improves overall traffic flow, enhances parking efficiency, and offers a practical solution to address the challenges associated with parking in urban areas. The below Figure 9-4 shows the view for the end users.

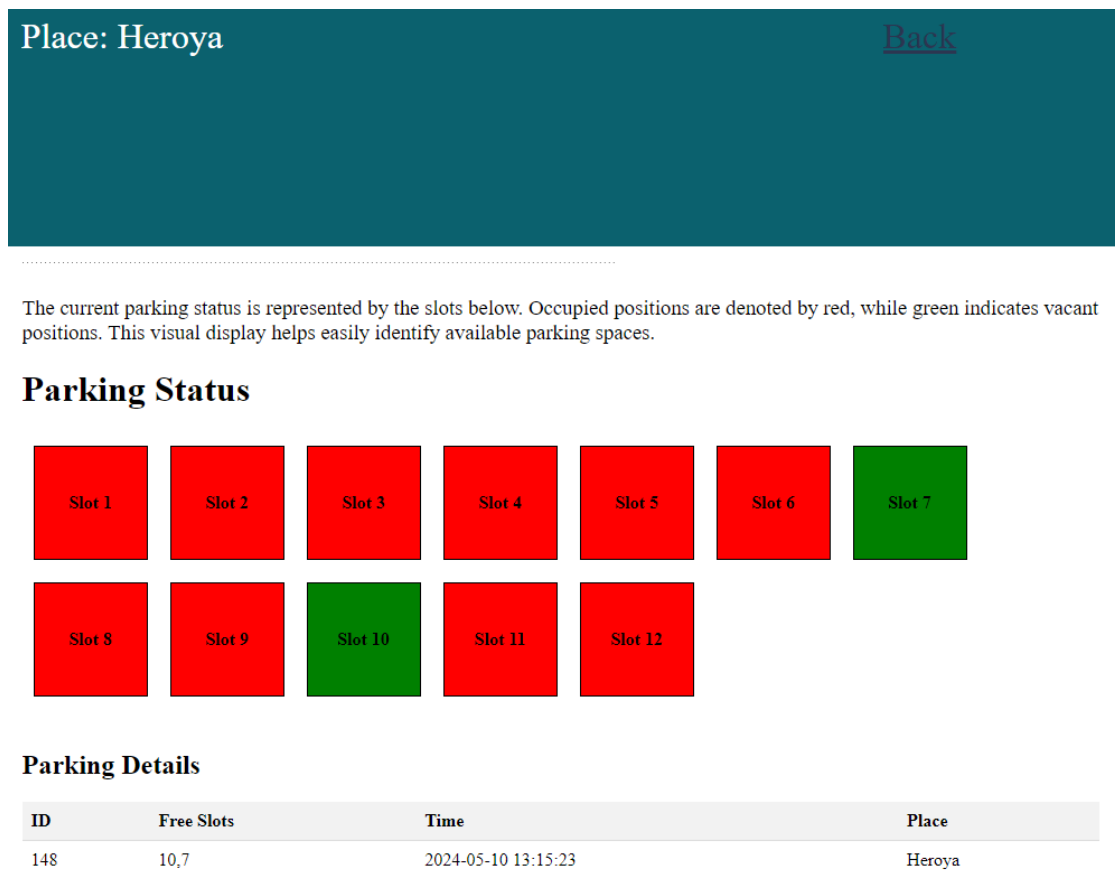


Figure 9-4 - Parking Slot Real-Time view of Smart Parking System in Website.

# 10 Implementation of the Prototype to Industrial Application

The successful implementation of IoT image processing in industrial applications requires two critical factors: durability and efficient processing speed. Industrial environments can be demanding, necessitating a robust system capable of withstanding harsh conditions. Additionally, the system must possess fast and reliable image processing capabilities to meet the high-speed requirements of industrial operations. By ensuring durability and optimized processing speed, the IoT image processing application can effectively contribute to improved performance and productivity in industrial settings.

## 10.1 Challenges Using Raspberry Pi in Industrial Development.

As Raspberry Pi has low processing power, it has some challenges regarding industrial development. They are [22]

1. **Processing Power:** While Raspberry Pi boards are capable devices, they have limited processing power compared to more powerful microprocessors or dedicated AI hardware. Image processing tasks, especially those involving complex algorithms or high-resolution images, can be computationally intensive and may exceed the capabilities of Raspberry Pi boards, resulting in slower performance or limited functionality.
2. **Memory Constraints:** Raspberry Pi boards typically have limited memory (RAM), which can also pose challenges for image processing applications. Processing large images or handling multiple images simultaneously can quickly consume available memory, leading to performance issues or even crashing of the application.
3. **Limited GPU Capability:** While some Raspberry Pi models have an integrated GPU, the GPU's capabilities are primarily designed for graphics acceleration and multimedia processing rather than general-purpose computing tasks. While it can assist in certain image processing tasks, it may not provide the level of performance or specialized functionality required for more advanced image processing algorithms.
4. **Power and Thermal Constraints:** Raspberry Pi boards are designed to be low-power devices, making them suitable for many IoT applications. However, image processing tasks can be power-hungry and generate significant heat, potentially leading to thermal throttling or instability if not properly managed. [23]

Despite these challenges, Raspberry Pi boards are still used for certain IoT image processing applications, especially those with lower computational demands or where real-time processing is not critical. Additionally, optimizing algorithms, using efficient libraries and frameworks, and leveraging hardware acceleration techniques can help improve the performance of image processing tasks on Raspberry Pi boards. For more demanding image processing applications, alternative microprocessors or dedicated AI hardware with higher processing power and specialized capabilities may be more suitable.

## 10.2 Why Nvidia Jetson is a Good Solution for This?

NVIDIA Jetson is a family of embedded AI computing platforms designed to accelerate AI and computer vision applications. Specifically, the Jetson platform, including Jetson Nano, Jetson Xavier NX, and Jetson AGX Xavier, offers powerful capabilities for image processing in IoT applications. Here's how Jetson works and its key capabilities: [24]

1. **High-Performance GPUs:** Jetson devices are equipped with NVIDIA GPUs, specifically designed for parallel processing, and accelerating AI workloads. These GPUs provide significant computational power, enabling efficient execution of image processing algorithms, including deep learning and computer vision tasks.
2. **AI Acceleration:** Jetson platforms incorporate hardware acceleration for AI tasks, such as tensor processing units (TPUs) or deep learning accelerators (DLAs). These dedicated hardware components enhance the performance of neural network inference, enabling faster and more efficient execution of image processing algorithms.
3. **CUDA and cuDNN Support:** Jetson devices support NVIDIA's CUDA parallel computing platform and the cuDNN deep neural network library. These tools provide a software framework for developers to optimize and accelerate their image processing algorithms using parallel computing techniques and GPU acceleration.
4. **JetPack SDK:** NVIDIA provides the JetPack Software Development Kit (SDK) specifically tailored for Jetson devices. JetPack includes a comprehensive set of software libraries, tools, and frameworks, including CUDA, cuDNN, TensorRT, and VisionWorks. These components greatly simplify the development and deployment of image processing applications on Jetson platforms.
5. **Connectivity and I/O:** Jetson devices offer a range of connectivity options, including Ethernet, USB, HDMI, and GPIO, allowing seamless integration with various IoT devices and sensors. This makes it easier to interface with cameras, capture video streams, and process images in real-time for IoT image processing applications.
6. **Power and Thermal Efficiency:** Jetson platforms are designed to balance performance and power efficiency. They feature power-efficient architectures and optimized thermal designs to handle the computational demands of image processing while maintaining stable operation.

These capabilities make NVIDIA Jetson an excellent choice for image processing in IoT applications. It can handle complex algorithms, real-time processing, and high-resolution images more efficiently compared to platforms like Raspberry Pi, making it well-suited for demanding computer vision tasks in IoT environments.

## 10.3 Jetson TX2 Module – Recommended Module

Jetson TX2 is the fastest, most power-efficient embedded AI computing device. This 7.5-watt supercomputer on a module brings true AI computing at the edge. It's built around an NVIDIA Pascal™-family GPU and loaded with 8GB of memory and 59.7GB/s of memory bandwidth. It features a variety of standard hardware interfaces that make it easy to integrate it into a wide range of products and form factors. [25]

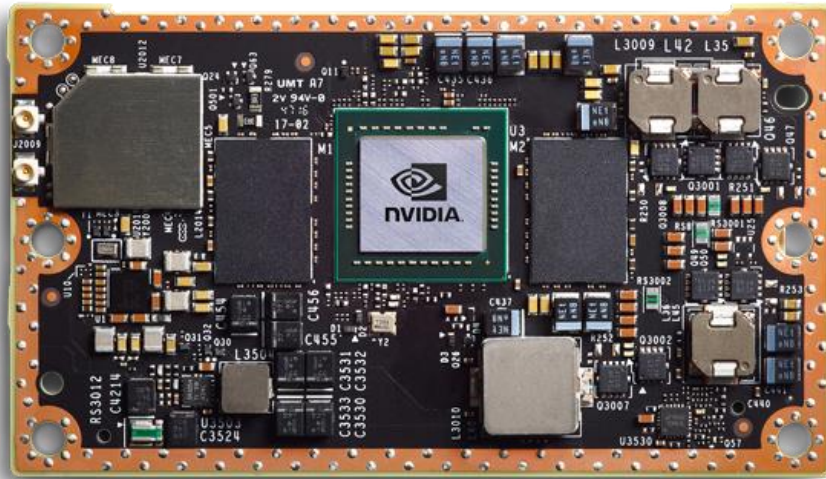


Figure 10-1 - Nvidia Jetson TX2 Module [25]

### 10.3.1 Technical Specifications

Table 10-1 - Jetson TX2 Specifications

<b>GPU</b>	256-core NVIDIA Pascal™ GPU architecture with 256 NVIDIA CUDA cores
<b>CPU</b>	Dual-Core NVIDIA Denver 2 64-Bit CPU Quad-Core ARM® Cortex®-A57 MPCore
<b>Memory</b>	8GB 128-bit LPDDR4 Memory 1866 MHz - 59.7 GB/s
<b>Storage</b>	32GB eMMC 5.1
<b>Power</b>	7.5W / 15W

## 10.4 Power Consumption and Battery Storage

A stable and uninterrupted power supply is of utmost importance in an image processing IoT traffic management system for several reasons. Firstly, the system relies on continuous power to operate its hardware components, including cameras, processors, and communication modules. Interruptions in power supply can result in system shutdowns, leading to a loss of real-time monitoring and traffic management capabilities. Secondly, power outages can disrupt data transmission, causing delays or even loss of critical information. Moreover, maintaining a reliable power source ensures the system's availability and functionality, allowing it to operate effectively and provide accurate traffic analysis and management. Therefore, a consistent and uninterrupted power supply is vital for ensuring the system's reliability, performance, and overall success.

### 10.4.1 Calculating Total Power Consumption

To calculate the required battery capacity to power the mentioned equipment for 20 hours, we need to determine the total power consumption of the devices over that period. Let's calculate it:

Nvidia Jetson: 15W \* 20 hours = 300 Wh  
 IM7600G-H 4G HAT: 1.5W \* 20 hours = 30 Wh  
 Camera: 1.5W \* 20 hours = 30 Wh

**Total power consumption: 300 Wh + 30 Wh + 30 Wh = 360 Wh**

For meeting this power requirement, a battery capacity of at least 360 watt-hours (Wh) to power the mentioned equipment for 20 hours, is necessary. This estimate assumes a continuous power consumption without any power-saving measures or variations in power usage. Additionally, it's advisable to select a battery with a slightly higher capacity to account for any inefficiencies or unexpected power fluctuations.

### 10.4.2 Selecting Suitable Power Battery Setups

When selecting a battery, consider factors such as capacity, portability, charging options, and any additional features that may be important for your specific use case. It's also advisable to check the power requirements and compatibility of the battery with the devices you intend to power. [26]

#### 10.4.2.1 Goal Zero Yeti 500X Portable Power Station



Figure 10-2 - Yeti 500X Portable Power Station [26]

### Specifications

Table 10-2 - Yeti 500X Portable Power Station

Product type	Portable Powerbank/Battery
Product series	Yeti
Color category	Grey, Green
Capacity (Ah)	46.8 A·h
Capacity (Wh)	505 W·h
Technology	Lithium-nickel-manganese-cobalt oxide
Battery voltage	10.8 V

## 10.5 Selecting a Suitable Highspeed Network Module

For NVIDIA Jetson processor in this IoT project, one option we can consider is the Quectel RM500Q-AE 5G module. It is a high-speed 5G module that provides reliable and fast connectivity for IoT applications. Here are some key features of the Quectel RM500Q-AE: [27]

1. **5G Connectivity:** The supports both standalone (SA) and non-standalone (NSA) modes, providing compatibility with different 5G network deployments. It offers high-speed data transfer rates, low latency, and enhanced network capacity.
2. **Multi-Mode Support:** The module supports various cellular networks, including 5G, LTE-A, LTE, and WCDMA, ensuring reliable connectivity even in areas without 5G coverage. It allows for seamless fallback to 4G or 3G networks when 5G is not available.
3. **GNSS Positioning:** The Quectel RM500Q-AE includes integrated GNSS (Global Navigation Satellite System) positioning capabilities, supporting GPS, GLONASS, BeiDou, and Galileo satellite systems. This enables accurate positioning and location-based services in your IoT project.
4. **Compact Form Factor:** The module is designed with a compact size, making it suitable for space constrained IoT applications. It can easily be integrated into your device or system alongside the NVIDIA Jetson processor.
5. **Industrial-Grade Reliability:** The Quectel RM500Q-AE is built to withstand harsh environmental conditions and operates reliably in industrial environments. It offers features such as temperature resistance, shock resistance, and extended lifespan.

Before integration Remember to check the compatibility and interface requirements of the 5G module with the NVIDIA Jetson processor to ensure seamless integration. Additionally, it's important to consider the availability and network compatibility of 5G services in your target deployment area.

### 10.5.1 Quectel RM500Q-AE 5G M.2 Module



Figure 10-3 -Quectel RM500Q-AE 5G M.2 Module [19]

## Hardware Specifications of Quectel RM500Q-AE 5G M.2 Module

Table 10-3 – Technical Specifications of Quectel RM500Q-AE 5G M.2 Module

Module	5G Sub-6GHz Module
5G NR NSA/SA	n1/n2/n3/n5/n7/n8/n12/n20/n25/n28/n38/n40/n41/n48/n66/n71/n77/n78/n79
LTE-FDD	B1/B2/B3/B4/B5/B7/B8/B12(B17)/B13/B14/B18/B19/B20/B25/B26/B28/B29/B30/B32/B66/B71
LTE-TDD	B34/B38/B39/B40/B41/B42/B43/B48
LTE LAA	B46 (only support 2 × 2 MIMO)
WCDMA	B1/8
GSM	B1/B2/B3/B4/B5/B6/B8/B19
GNSS	GPS/GLONASS/BDS/Galileo
Max. Download Speed	2.5 Gbps
Max. Upload Speed	900 Mbps
Form Factor	M.2
Dimensions	52.0 × 30.0 × 2.3 mm
Temperature Range	-40°C to +85°C

## 10.6 Cost Estimation of Initiating the Project

To provide a cost estimation for mounting cameras in streets in Norway, it's important to consider that costs can vary depending on several factors such as the number of cameras, camera quality, installation requirements, ongoing maintenance, and other specific project needs. Additionally, real-time pricing information is specific from vendor to vendors in Norway. However, below with a general breakdown of potential costs involved in such a project:

- **Camera Equipment:** The cost of cameras can vary widely based on their features, resolution, and capabilities. High-quality surveillance cameras can range from a few hundred to several thousand Norwegian Krone (NOK) per camera.
- **Hardware Installation:** The installation cost can involve labor charges for mounting the cameras, running cables, and ensuring proper connections. The complexity of the installation, such as the number of cameras and the distance between them, can impact the overall cost. It's recommended to consult with local vendors or installation professionals to obtain accurate quotes.



- **Network Infrastructure:** Depending on the project requirements, you may need to invest in network provider to support the computer system. This can include LTE modules network switches, routers, and cabling. The cost will depend on the scale and complexity of the network infrastructure needed.
- **Storage and Recording:** The size of the storage required will depend on factors such as the number of records, processor processing speed, and retention period.
- **Power and Connectivity:** Ensure that you include the cost of providing power to the Edge units, which may involve electrical work or the installation of power sources. Additionally, consider the cost of internet connectivity for remote monitoring and access to the camera system.
- **Maintenance and Support:** Ongoing maintenance, software updates, and technical support may be necessary. It's advisable to budget for periodic maintenance and potential technical assistance from vendors or service providers.

### 10.6.1 Estimating Budget for Traffic Management System

The cost of mounting cameras in streets in Porsgrunn commune depends on the number of locations where vehicles can access the area. By referring to the map of toll counters in Porsgrunn city, we can assume these locations as points to measure traffic counts. The cost estimation for mounting cameras will take into consideration the number of cameras required at these points, installation costs, network infrastructure, storage, power requirements, and ongoing maintenance. It is recommended to consult with local suppliers or experts in Porsgrunn for a more accurate cost estimation tailored to your specific project and location. As the below toll map in Figure 10-4 - Porsgrunn Toll Points [28] we can take the number of entry points to the city.

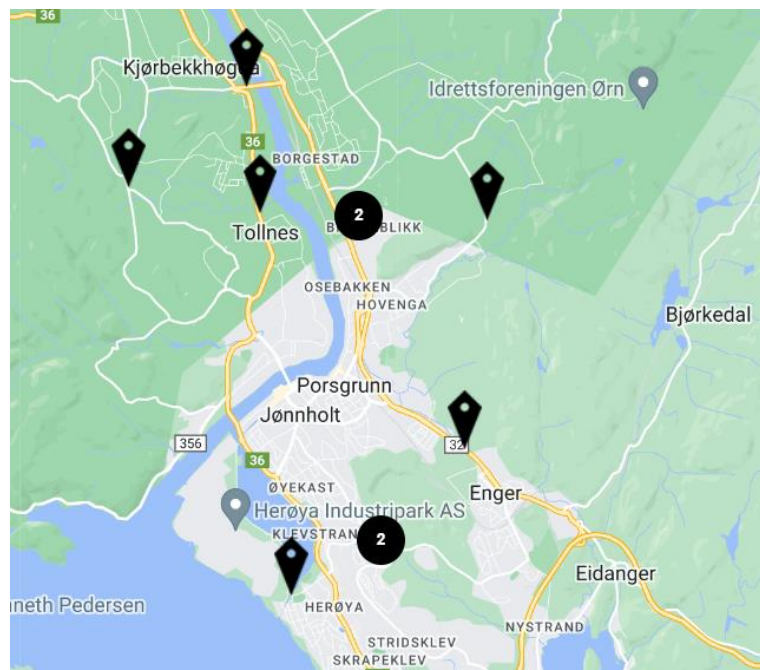


Figure 10-4 - Porsgrunn Toll Points [28]

Based on the details the hardware and software estimation can be calculated as below.

**Per unit calculation-**

The Table 10-4 - Per Unit Calculation for Unit and Services demonstrates how much it can cost for a unit installation.

Table 10-4 - Per Unit Calculation for Unit and Services

Unit & services	Source	Price (NOK)
Nvidia Jetson TX2 module	<a href="#">Link</a>	6637.5
Quectel RM500Q-AE 5G M.2 Module	<a href="#">Link</a>	3,237.16
Goal Zero Yeti 500X Portable Power Station	<a href="#">Link</a>	4649
Camera	<a href="#">Link</a>	2799
6MX12M Traffic Light Pole	<a href="#">Link</a>	5000
Installation materials and Manpower		7000
<b>Total</b>		<b>29322.66</b>

**Software Costs –**

The Table 10-5 - Server and Software Costs for the Project demonstrate how it can be costs for the servers and installations.

Table 10-5 - Server and Software Costs for the Project

Services	Source	Price (NOK)
Webserver( <a href="https://www.000webhost.com/">https://www.000webhost.com/</a> )	<a href="#">Link</a>	413.71
Tableau Server	<a href="#">Link</a>	10498.67
AWS Infrastructure	<a href="#">Link</a>	2,973.86
<b>Total</b>		<b>13886.239</b>

From the map Figure 10-4 - Porsgrunn Toll Points there are 10 points which vehicles enter to the Porsgrunn city. Therefore, for this project the estimated cost can be calculated as below

$$\begin{aligned}
 \text{Total Cost (NOK)} &= \text{No of entry points} * \text{Per unit cost} + \text{Software cost} \\
 &= 10 * 29,322.66 + 13,886.239 \\
 &= \mathbf{307,112.839}
 \end{aligned}$$

Therefore, an approximate cost of **307,113** will be costs for the project.

**10.6.2 Estimating Budget for Smart Parking System**

Parking depends on the number of parking slots. but for a unit we can estimate the costs as below

$$\begin{aligned}
 \text{Total Cost} &= \text{Per Unit} * \text{No of parking slots} + \text{Software costs} \\
 &= 29,322.66 * n + 13886.239
 \end{aligned}$$

Therefore, the project will cost **approximately 29,322.66 n + 13886.239 (where n is the number of parking slots).**

# 11 Discussion

Real-time data updating plays a crucial role in enhancing decision-making in traffic management. By providing up-to-the-minute information on traffic conditions, it enables traffic managers to make informed and timely decisions. This real-time data includes details on congestion levels, accidents, road closures, and other relevant information that directly impacts traffic flow. In the project, microprocessors are utilized for image recognition, enabling the system to accurately detect and analyze vehicles in real time. These microprocessors process the incoming image data, applying image recognition algorithms to identify vehicles, track their movements, and extract relevant data for traffic analysis.

To ensure efficient data management, servers are set up for the database. These servers store the collected data, including real-time traffic information and parking occupancy data. The database is continuously updated with fresh data, allowing for accurate and timely analysis. Additionally, AWS instances are employed for hosting the Tableau server and Tableau dashboards. Tableau is a powerful data visualization and analysis platform that provides interactive and insightful visual representations of the collected traffic and parking data. By leveraging AWS instances, the project ensures scalability, reliability, and secure access to the Tableau server and dashboards. The Tableau dashboards allow traffic managers and stakeholders to analyze the real-time data and gain valuable insights into traffic patterns, parking occupancy rates, and other relevant metrics. This data-driven approach empowers decision-makers to implement effective strategies for traffic management, such as adjusting signal timings, optimizing parking allocation, and deploying resources efficiently.

In summary, the project combines microprocessors for image recognition, server setups for database management, and AWS instances for hosting the Tableau server and dashboards. This technical infrastructure enables the collection, processing, and visualization of real-time traffic and parking data, facilitating informed decision-making, and enhancing overall traffic management effectiveness.

## 11.1 Further Implementation

For further implementation of the project, several key aspects can be considered:

1. **Image Recognition Optimization:** Explore advanced image recognition algorithms and techniques to improve the accuracy and efficiency of vehicle detection and analysis. This could involve deep learning models or edge computing approaches to enhance real-time processing capabilities.
2. **Data Integration:** Integrate data from various sources beyond traffic conditions, such as weather information, public transportation schedules, and event data. This broader dataset can provide a more comprehensive understanding of the traffic ecosystem and enable better decision-making.
3. **Predictive Analytics:** Implement predictive analytics models that leverage historical data and real-time inputs to forecast traffic patterns and anticipate congestion hotspots. This proactive approach can help traffic managers take preventive measures and optimize traffic flow in advance.

4. **Mobile Application Development:** Create a user-friendly mobile application that allows drivers to access real-time traffic updates, parking availability, and personalized navigation suggestions. This empowers individuals to make informed decisions and contributes to overall traffic management efforts.
5. **Data Security and Privacy:** Implement robust security measures to protect the collected data, ensuring compliance with privacy regulations. This includes encryption protocols, access controls, and regular security audits to safeguard sensitive information and maintain public trust.
6. **Machine Learning for Anomaly Detection:** Utilize machine learning algorithms to identify anomalous traffic behavior, such as accidents or road closures, in real-time. By automatically detecting and alerting traffic managers to these events, response times can be improved, minimizing disruptions and enhancing overall traffic management efficiency.
7. **Integration with Intelligent Transportation Systems (ITS):** Collaborate with existing ITS infrastructure, such as traffic signal control systems or parking guidance systems, to create a more comprehensive and interconnected traffic management ecosystem. This integration allows for coordinated actions and optimized resource allocation across different systems.

By considering these aspects, the project can be further enhanced to provide even more accurate, efficient, and comprehensive real-time data analysis for traffic management.

## 12 Summary

The provision of real-time parking data and traffic data holds immense importance for the community, offering several key benefits that positively impact individuals and the overall urban environment. Real-time traffic data significantly improves traffic flow. By providing up-to-date information on road conditions, congestion, and accidents, commuters can make informed decisions about their routes. This helps them avoid congested areas and choose more efficient paths, ultimately reducing travel time, frustration, and fuel consumption. Smoother traffic flow not only benefits individual drivers but also contributes to overall road safety and efficiency. Then, real-time parking data enables drivers to quickly and conveniently locate available parking spaces. By accessing information on parking availability and occupancy rates, individuals can make informed choices, reducing the time spent searching for parking. This leads to decreased congestion in parking lots, improved parking utilization, and a more efficient use of urban space.

Furthermore, the availability of real-time data contributes to reduced environmental impact. By minimizing the time spent searching for parking and optimizing traffic flow, fuel consumption and greenhouse gas emissions are reduced. This environmentally friendly approach aligns with sustainability goals and helps create a healthier and more sustainable community. Real-time data also plays a significant role in urban planning. Accurate and up-to-date traffic and parking data provide valuable insights for urban planners and policymakers. It enables them to identify areas of high demand, assess transportation infrastructure needs, and make data-driven decisions regarding traffic management, parking allocation, and public transportation improvements. This leads to more efficient and well-planned urban environments, benefiting residents and visitors alike. Overall, the availability of real-time parking data and traffic data improves transportation efficiency, reduces congestion, and enhances the overall quality of life within the community. The developed image recognition vehicle counting system and smart parking system offer significant advancements in traffic management and parking efficiency. These systems utilize cutting-edge image recognition technology to accurately count and categorize different types of vehicles, providing real-time data for improved traffic planning and infrastructure development. By integrating machine learning algorithms and predictive modeling, further enhancements can be achieved, ensuring scalability and adaptability to diverse environments. While challenges such as varying lighting conditions and occlusions may arise, the benefits of these systems include reduced congestion, optimized parking utilization, enhanced urban planning, and an overall improved transportation experience for users. The potential for future developments lies in refining accuracy, scalability, and incorporating real-time data analysis to enable proactive traffic management strategies.

# 13 References

- [1] A. P. Gulati, «Vehicle Detection and Counting System using OpenCV,» 31 12 2021. [Internett]. Available: <https://www.analyticsvidhya.com/blog/2021/12/vehicle-detection-and-counting-system-using-opencv/>.
- [2] H. L. Huansheng Song, «Vision-based vehicle detection and counting system using deep learning in highway scenes,» 12 2019. [Internett]. Available: [https://www.researchgate.net/publication/338251870\\_Vision-based\\_vehicle\\_detection\\_and\\_counting\\_system\\_using\\_deep\\_learning\\_in\\_highway\\_scenes](https://www.researchgate.net/publication/338251870_Vision-based_vehicle_detection_and_counting_system_using_deep_learning_in_highway_scenes).
- [3] B. Wolford, «Editor in Chief, GDPR EU,» GDPR.eu, 2024.
- [4] S. Garfnkel, «De-Identifying Government Datasets:Techniques and Governance,» i *National Institute of Standards and Technology*.
- [5] R. Koch, «Managing Editor, GDPR EU,» GDPR.EU, 2020. [Internett]. Available: <https://gdpr.eu/working-remotely-data-security/>. [Funnet 15 4 2024].
- [6] «Video-surveillance,» EUROPEAN DATA PROTECTION SUPERVISOR, [Internett]. Available: [https://www.edps.europa.eu/data-protection/data-protection/reference-library/video-surveillance\\_en](https://www.edps.europa.eu/data-protection/data-protection/reference-library/video-surveillance_en).
- [7] M. Roza, «CSA's Perspective on Cloud Risk Management,» 20 08 2020. [Internett]. Available: <https://cloudsecurityalliance.org/artifacts/csa-s-perspective-on-cloud-risk-management>.
- [8] ISO, «Information security, cybersecurity and privacy protection,» *Information security management systems*, 2022.
- [9] E. M. A. Abdelrahman Elesawy, «A Detailed Comparative Analysis of You Only Look Once-Based Architectures for the Detection of Personal Protective Equipment on Construction Sites,» nr. 2024, 2023.
- [10] A. K. A. Oluwaseyi Ezekiel Olorunshola, «Comparative Study of Some Deep Learning Object Detection Algorithms: R-CNN, FAST R CNN, FASTER R-CNN, SSD, and YOLO,» NILE UNIVERSITY, Nigeria, 2023.
- [11] R. Kndu, «YOLO: Algorithm for Object Detection Explained,» 17 01 2023. [Internett]. Available: <https://www.v7labs.com/blog/yolo-object-detection>.
- [12] A. Acharya, 4 4 2024. [Internett]. Available: <https://encord.com/blog/yolo-object-detection-guide/#h5>.
- [13] S. D. Joseph Redmon, «You Only Look Once:Unified, Real-Time Object Detection,» University of Washington, Washington.

- [14] A. Burnes, «FrameView Performance and Power Benchmarking App,» nvidia, 20 10 2022. [Internet]. Available: <https://www.nvidia.com/en-us/geforce/news/nvidia-frameview-power-and-performance-benchmarking-app-download/>.
- [15] sciotex, «GPU vs. FPGA vs. CPU: A Comparative Analysis for Image Processing in AI and Traditional Machine Vision,» sciotex.
- [16] N. Foster, «GPU vs. CPU for Image Processing: Which One is Better?,» 06 10 2022. [Internet]. Available: <https://www.acecloudhosting.com/blog/gpu-vs-cpu-for-image-processing/>.
- [17] T. Brant, «Raspberry Pi 4 Review,» PCMag UK , 7 8 2019. [Internet]. Available: <https://uk.pcmag.com/desktop-pcs/122009/raspberry-pi-4>.
- [18] R. Pi, «Raspberry Pi 4 Tech Specs,» Raspberry Pi , [Internet]. Available: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>.
- [19] waveshare, «SIM7600G-H 4G HAT (B) for Raspberry Pi, LTE Cat-4 4G / 3G / 2G Support, GNSS Positioning, Global Band,» waveshare, [Internet]. Available: <https://www.waveshare.com/sim7600g-h-4g-hat-b.htm>.
- [20] talentcell, «TalentCell 12V Lithium ion Battery PB120B2, Rec PB120B2,» talentcell, [Internet]. Available: <https://talentcell.com/lithium-ion-battery/12v/pb120b2.html>.
- [21] Anker, «Anker 737 Power Bank (PowerCore 24K),» Anker, [Internet]. Available: <https://www.anker.com/products/a1289>.
- [22] engtong, «Challenges in Using Pi 4 in Industrial Projects And The Solution,» cytron, 14 3 2023. [Internet]. Available: <https://www.cytron.io/tutorial/challenges-in-using-pi-4-in-industrial-projects-and-the-solution>.
- [23] Maida, «The Challenges and Opportunities of Raspberry Pi in Industrial Automation: Exploring Applications,» Maida, 11 4 2024. [Internet]. Available: <https://www.raspberrypibox.com/the-challenges-and-opportunities-of-raspberry-pi-in-industrial-automation-exploring-applications/>.
- [24] F. Serzhenko, «Jetson image processing for camera applications,» Nvidia, [Internet]. Available: <https://www.fastcompression.com/blog/jetson-image-processing.htm>.
- [25] NVIDIA, «NVIDIA Jetson TX2 Enables AI at the Edge,» NVIDIA , 7 3 2017. [Internet]. Available: <https://forums.developer.nvidia.com/t/performance-question/50272>.
- [26] dustin, «Goal Zero Yeti 500X Portable Power Station,» Goal Zero, [Internet]. Available: <https://www.dustin.no/product/5011179238/yeti-500x-portable-power-station?tab=specification&scrollToTab=true>.
- [27] Quectel, «Quectel RM500Q-AE IoT/eMBB-Optimized 5G Sub-6 GHz M.2 Module,» Quectel, [Internet]. Available: [https://www.quectel.com/wp-content/uploads/2021/03/Quectel\\_RM500Q-AE\\_5G\\_Specification\\_V1.1.pdf](https://www.quectel.com/wp-content/uploads/2021/03/Quectel_RM500Q-AE_5G_Specification_V1.1.pdf).

[28] fremtindservice, «Use the toll calculator - avoid surprises,» [Internett]. Available:  
<https://fremtindservice.no/private/toll-calculator/>.

[29] <https://www.scandinavianphoto.no>, «HD widescreen video with 720p quality and clear sound,» Microsoft, [Internett]. Available:  
<https://www.scandinavianphoto.no/microsoft/lifecam-cinema-1003038>.



# 14 Appendices

Appendix A - FMH606 Master's Thesis

# Appendix A - FMH606 Master's Thesis

**Title:** Designing, Implementation and Testing Smart Cities Use Cases with respect to practical Internet of Things Applications

**USN supervisor:** Hans-Petter Halvorsen

**External partner:** Porsgrunn kommune (Porsgrunn municipality), Altibox

## **Task background:**

Since the autumn of 2018, the Altibox and Altibox partnership has expanded the LoRaWAN Sensor Network in Norway and currently has coverage for more than 1,000,000 households in 100 municipalities. The LoRaWAN Sensor Network is a natural extension of the fiber network with major synergies of established infrastructure and the development continues.

Altibox and Partners now offer IoT Access as a commercial service, so that more people can use the Sensor Network for their own sensors.

More information: <https://www.altibox.no/iot/>

Porsgrunn kommune (Porsgrunn municipality) is one of the main users of the LoRaWAN Sensor Network by Altibox and they are taking the next step to be a so-called Smart City where they use Internet of Things (IoT) solutions as part of their services for the residents.

More information: <https://www.porsgrunn.kommune.no>

## **Task description:**

Examples of activities that should be performed:

- Get an overview of LoRaWAN, NB-IoT, vision systems and other technologies relevant for smart cities applications.
- How is Porsgrunn municipality focusing on smart cities aspects today? Get an overview. Existing use cases? Some examples may be water temperature monitoring, waste management, air pollution, traffic control and surveillance, parking, etc. Literature study: What is published online regarding this?
- Work with Porsgrunn municipality to find and explore relevant user cases for the residents in Porsgrunn municipality related to this work.
- The user cases should demonstrate the features and possibilities in context of the new Smart Cities era in Porsgrunn municipality. Some examples of use cases may be rainwater and measure and monitoring of storm water from wells, vehicle counting, etc.
- Plan, design, implement and test one or more selected user case(s) that will benefit the residents in Porsgrunn municipality.
- Collaboration with other ongoing LoRaWAN Relay project that are carried out in parallel.
- Microsoft Teams and other relevant systems should be used during project planning and development.
- The system should be properly documented in form of a technical report.

More project details and activities will be discussed when the project starts. You will have great opportunities to influence the content of the project in collaboration with Porsgrunn kommune (Porsgrunn municipality).

**Student category:** IIA

**Is the task suitable for online students (not present at the campus)?** Yes, but is advisable that the student can be present in Porsgrunn and Porsgrunn kommune (Porsgrunn municipality) for some parts of the work.


**Practical arrangements:**

**Supervision:**

The student is entitled to 15-20 hours of supervision. This includes necessary time for the supervisor to prepare for supervision meetings (reading material to be discussed, etc).

**Signatures:**

Supervisor (date and signature):

  
2024.02.02

Student



Gaveen Shamila Ranabahu

2024.02.02

Date