INTERNET OF THINGS

IBM Naan Mudhalvan Phase-5

Project Tittle: TRAFFIC MANAGEMENT

Project Objectives:

The Primary Objective of this project is to implement an IoT-based traffic management system using cameras and smart sensors. The key goals include:

- Real-time monitoring of traffic flow and congestion.
- Automated detection of traffic violations and accidents.
- Optimized traffic signal control for efficient vehicle movement.
- Data-driven insights for future infrastructure planning.

IoT Device Setup:

1. SENSOR/CAMERA DATA COLLECTION: Traffic controlling sensors/cameras collect data on the road. It collects the vehicle capacity on the road. These sensors continuously measure the occupation of vehicles on the road.

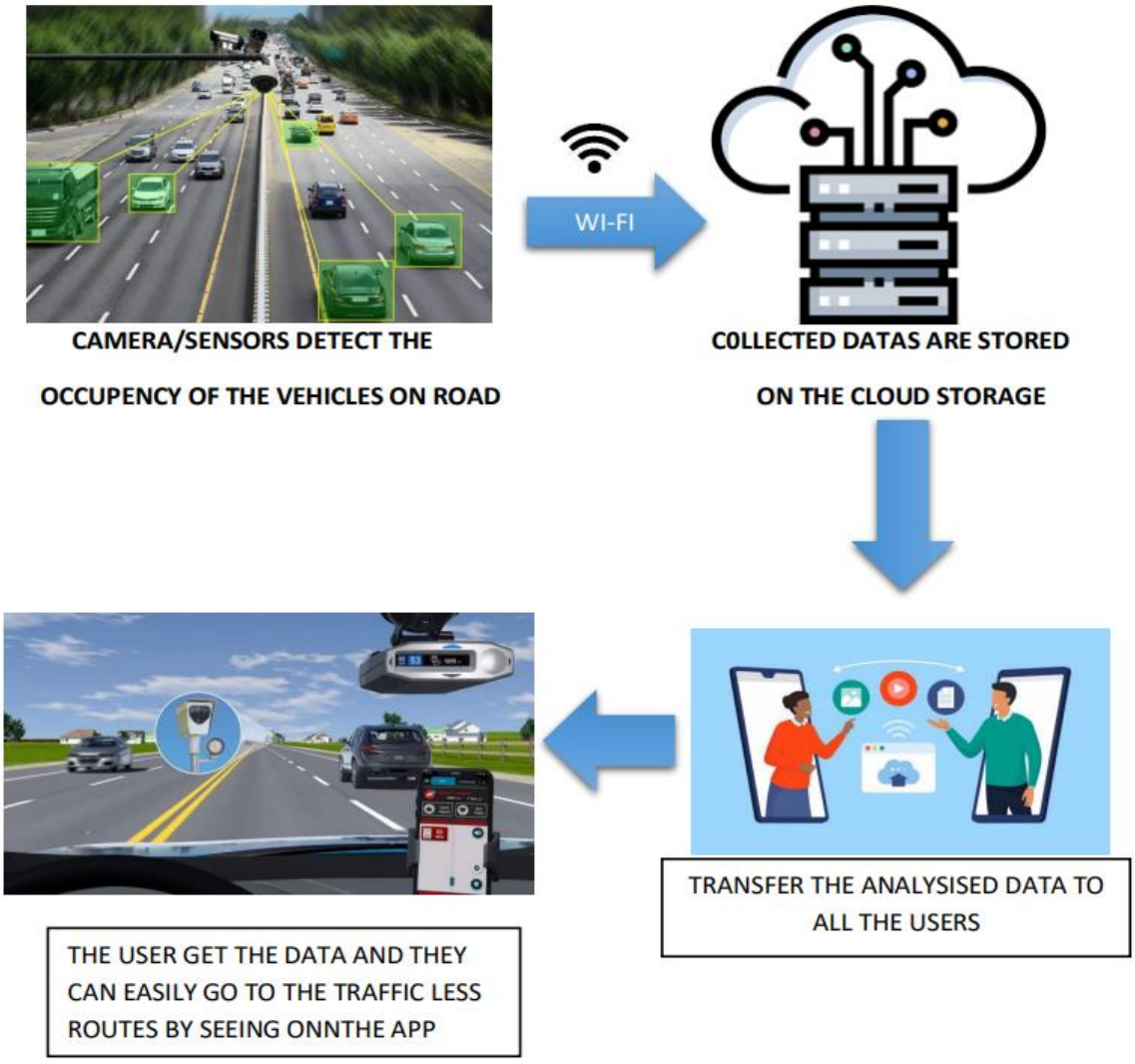
2. DATA TRANSMISSION: The collected sensor data is transmitted in Real-Time to the IoT platform using wireless communication protocols such as Wi-Fi, Bluetooth, or cellular networks to cloud storage.

3. DATA STORED IN CLOUD: Cloud Storage is a mode of computer data storage in which digital data is stored on servers in off-site locations. The servers are maintained by a third-party provider who is responsible for hosting, managing, and securing data stored on its infrastructure. The provider ensures that data on its servers is always accessible via public or private internet connections.

Cloud Storage enables organizations to store, access, and maintain data so that they do not need to own and operate their own data centres, moving expenses from a capital expenditure model to operational. Cloud Storage is scalable, allowing organizations to expand or reduce their data footprint depending on need.

4. DATA ANALYSIS: Data analysis algorithms are applied to the stored data to assess the availability of free routes for journey. These algorithms can compare the collected data against the vehicles occupying on the road. If the occupying space is low, the system continues monitoring.

5. ALERT GENERATION: If the analysis indicates poor capacity on the road, an alert is generated. This could trigger various actions such as sending notifications to users, local authorities, or facility managers.



Platform Development:

In continuing the build of traffic management platform, there is a need to consider various aspects such as User Interface, Data Processing, Real-Time Updates, and possibly integration with external systems.

Project Outline:

1. Requirement Gathering:

- Defining the specific requirements of the traffic management platform.
- Identifying the user roles and their functionalities.
- Determining the scope of the platform (e.g., real-time monitoring, historical data analysis, user alerts).

2. Architecture Design:

- Designing the System Architecture to meet the project requirements.
- Considering a Scalable and Modular Architecture.
- A suitable Plan for Data Storage, Processing, and User Interfaces.

3. Frontend Development:

- The Frontend Development is done using
  - HTML
  - CSS
  - JavaScript

4. Backend Development:

- Choosing a Backend Language Node.js and a framework Express js
- Setting up a Database SQL for storing traffic data.

5. Real-time Updates:

- The Technologies like WebSocket is used for real-time updates.
- Several features were implemented such that it allow users to receive live Traffic Information.

6. Data Processing:

- The Algorithms are developed for processing traffic data.
- Machine Learning models are considered for predictive analysis.
- Data Cleansing and Validation Mechanisms are implemented.

7. Integration:

- These are Integrated with external APIs or systems for Additional Data Sources.
- The compatibility with existing traffic management infrastructure in ensured.

8. Security:

- Implementation of Authentication and Authorization Mechanisms.
- The sensitive data's especially if dealing with Traffic-Related Security Issues are Encrypted.

9. Testing:

- The Unit Testing for individual components are performed.
- The integration testing is done to ensure all parts of the system work together.
- Load testing for scalability.

10. Documentation:

- The comprehensive documentation for the project includes
  - Installation Guides
  - API Documentation
  - User Manuals.

11. Deployment:

- Deploying the platform on a web server or cloud infrastructure (e.g., AWS, Azure, Heroku).
- Setting up Continuous Integration/Continuous Deployment (CI/CD) pipelines if possible.

12. Monitoring and Maintenance:

- The Monitoring Tools are implemented to track System Performance.
- The alerts Set up for potential issues.

Project Structure:

1. Folder Structure:

Traffic-management/

├── public/

│ ├── index.html

│ ├── styles.css

│ ├── app.js

├── server.js

└── package.json

2. Dependencies:

- Express: 'npm install express'
- Socket.io (for real-time communication): 'npm install socket.io'

Frontend (public/index.html):

HTML:

<!DOCTYPE html>

<html lang="en">

<head>

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Traffic Management with Cameras</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <video id="video" width="640" height="480" autoplay></video>
    <canvas id="canvas" width="640" height="480"></canvas>
    <script src="https://cdn.socket.io/4.0.1/socket.io.min.js"></script>
    <script src="app.js"></script>
</body>
</html>
```

Frontend (public/styles.css):

CSS:

```
body {
    margin: 0;
    padding: 0;
}

video, canvas {
    display: block;
    margin: 20px auto;
}
```

Frontend (public/app.js):

Javascript:

```
const video = document.getElementById('video');
const canvas = document.getElementById('canvas');
const ctx = canvas.getContext('2d');
```

```
navigator.mediaDevices.getUserMedia({ video: true })
    .then((stream) => {
        video.srcObject = stream;
    })
    .catch((error) => {
        console.error('Error accessing camera:', error);
    });

video.addEventListener('play', () => {
    const socket = io();

    function processVideo() {
        ctx.drawImage(video, 0, 0, 640, 480);
        const imageData = ctx.getImageData(0, 0, 640, 480);
        socket.emit('videoFrame', imageData.data.buffer);
        requestAnimationFrame(processVideo);
    }
    processVideo();
});
```

Backend (server.js):

```
const express = require('express');
const http = require('http');
const socketIO = require('socket.io');
const app = express();
const server = http.createServer(app);
const io = socketIO(server);

app.use(express.static('public'));
```

```
io.on('connection', (socket) => {
  console.log('A user connected');

  socket.on('videoFrame', (data) => {
    // Process video frame data (you would perform object detection and analysis here)
    // For simplicity, let's just log the number of bytes received
    console.log('Received video frame:', data.byteLength);
  });
  socket.on('disconnect', () => {
    console.log('User disconnected');
  });
});
const port = process.env.PORT || 3000;
server.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```

Code implementation:

Configuring IoT devices to measure Traffic Level and developing a Python script to send collected data to a Data-Sharing platform involves several steps.

Build a Vehicle Detection System using OpenCV and Python:

The Computer Vision Library is used i.e.., OpenCV (version – 4.0.0) a lot in this implementation. Let's first import the required libraries and the modules.

Import Libraries

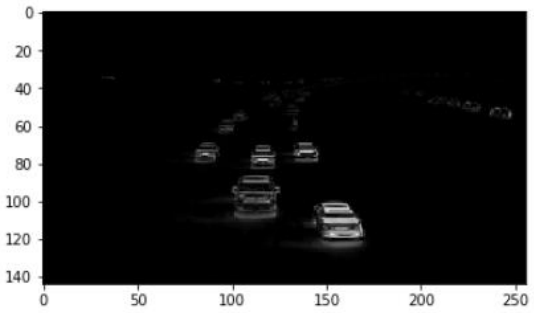
```
import os
import re
import cv2 # opencv library
```

```
import numpy as np
    from os.path import isfile, join
    import matplotlib.pyplot as plt
```

Import Video Frames and Data Exploration:

The folder named "frames" is inside the working directory. From that folder, the frames are imported and kept in a list and then for Data Exploration.

```
# convert the frames to grayscale
        grayA = cv2.cvtColor(col_images[i], cv2.COLOR_BGR2GRAY)
        grayB = cv2.cvtColor(col_images[i+1], cv2.COLOR_BGR2GRAY)
        # plot the image after frame differencing
        plt.imshow(cv2.absdiff(grayB, grayA), cmap = 'gray')
        plt.show()
```

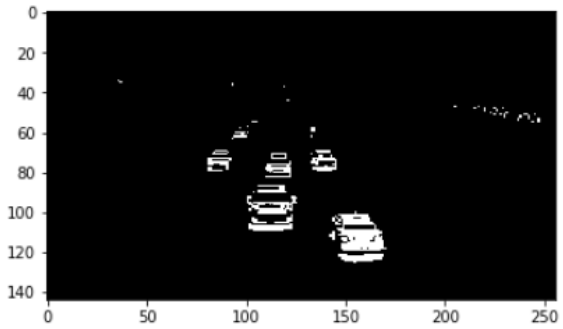


The moving objects in the 13th and 14th frames is clearly visible while everything else that was not moving has been subtracted out.

Image Pre-processing:

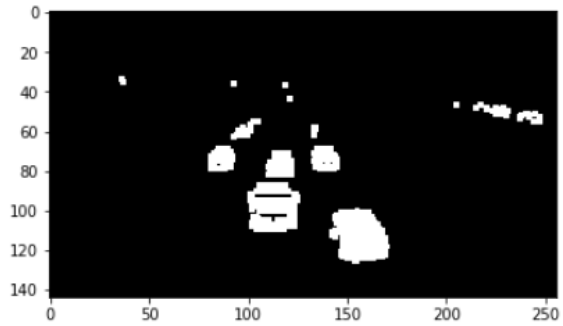
After applying thresholding to the above image:

```
diff_image = cv2.absdiff(grayB, grayA)
    # perform image thresholding
    ret, thresh = cv2.threshold(diff_image, 30, 255, cv2.THRESH_BINARY)
    # plot image after thresholding
    plt.imshow(thresh, cmap = 'gray')
    plt.show()
```



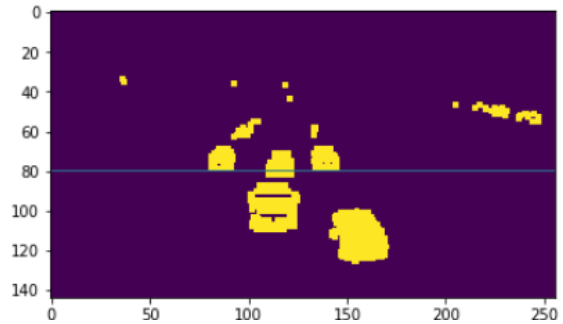
The moving objects (vehicles) look more promising and most of the noise (undesired white regions) are gone. However, the highlighted regions are a bit fragmented. So on applying image dilation over this image:

```
# apply image dilation
kernel = np.ones((3,3),np.uint8)
dilated = cv2.dilate(thresh,kernel,iterations = 1)
# plot dilated image
plt.imshow(dilated, cmap = 'gray')
plt.show()
```



The moving objects have more solid highlighted regions. Hopefully, the number of contours for every object in the frame will not be more than three.

```
# plot vehicle detection zone
plt.imshow(dilated)
cv2.line(dilated, (0, 80),(256,80),(100, 0, 0))
plt.show()
```



The area below the horizontal line y = 80 is the vehicle detection zone. The detection of any movement happens in this zone only.

Finding the contours in the detection zone of the above frame:

```
# find contours
contours,hierarchy=cv2.findContours(thresh.copy(),cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
```

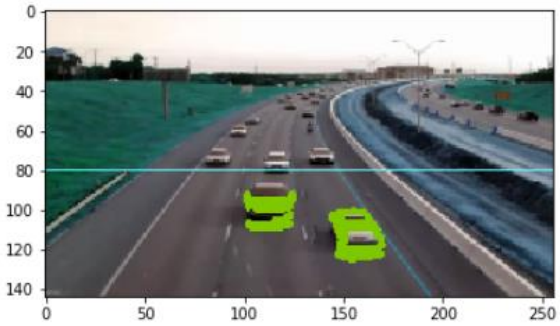
The code above finds all the contours in the entire image and keeps them in the variable contours. Since There is a need to find only those contours that are present in the detection zone, a couple of checks on the discovered contours is applied.

The first check is whether the top-left y-coordinate of the contour should be >= 80 (I am including one more check, x-coordinate <= 200). The other check is that the area of the contour should be >= 25. You can find the contour area with the help of the cv2.contourArea( ) function.

```
valid_cntrs = []
for i,cntr in enumerate(contours):
   x,y,w,h = cv2.boundingRect(cntr)
   if (x <= 200) & (y >= 80) & (cv2.contourArea(cntr) >= 25):
      valid_cntrs.append(cntr)
# count of discovered contours
len(valid_cntrs)
```

The contours along with the Original Frame:

```
dmy = col_images[13].copy()
cv2.drawContours(dmy, valid_cntrs, -1, (127,200,0), 2)
cv2.line(dmy, (0, 80),(256,80),(100, 255, 255))
plt.imshow(dmy)
plt.show()
```



Contours of only those vehicles that are inside the detection zone are visible.

Vehicle Detection in Videos:

It's time to apply the same image transformations and pre-processing operations on all the frames and find the desired contours. Just to reiterate, we will follow the below steps:

1. Apply frame differencing on every pair of consecutive frames
2. Apply image thresholding on the output image of the previous step
3. Perform image dilation on the output image of the previous step
4. Find contours in the output image of the previous step
5. Shortlist contours appearing in the detection zone
6. Save frames along with the final contours

```
# kernel for image dilation
kernel = np.ones((4,4),np.uint8)

# font style
font = cv2.FONT_HERSHEY_SIMPLEX

# directory to save the ouput frames
pathIn = "contour_frames_3/"

for i in range(len(col_images)-1):

    # frame differencing
    grayA = cv2.cvtColor(col_images[i], cv2.COLOR_BGR2GRAY)
    grayB = cv2.cvtColor(col_images[i+1], cv2.COLOR_BGR2GRAY)
    diff_image = cv2.absdiff(grayB, grayA)

    # image thresholding
    ret, thresh = cv2.threshold(diff_image, 30, 255, cv2.THRESH_BINARY)

    # image dilation
    dilated = cv2.dilate(thresh,kernel,iterations = 1)

    # find contours

    Contours,hierarchy=cv2.findContours(dilated.copy(),cv2.RETR_TREE,cv2.CHAI
    N_APPROX_NONE)
    # shortlist contours appearing in the detection zone
    valid_cntrs = []
    for cntr in contours:
```

```
            x,y,w,h = cv2.boundingRect(cntr)
            if (x <= 200) & (y >= 80) & (cv2.contourArea(cntr) >= 25):
                if (y >= 90) & (cv2.contourArea(cntr) < 40):
                    break
                valid_cntrs.append(cntr)
        # add contours to original frames
        dmy = col_images[i].copy()
        cv2.drawContours(dmy, valid_cntrs, -1, (127,200,0), 2)
        cv2.putText(dmy, "vehicles detected: " + str(len(valid_cntrs)), (55, 15), font, 0.6, (0,
180, 0), 2)
        cv2.line(dmy, (0, 80),(256,80),(100, 255, 255))
        cv2.imwrite(pathIn+str(i)+'.png',dmy)
```

Video Preparation:

The contours are added for all the moving vehicles in all the frames.

```
# specify video name
pathOut = 'vehicle_detection_v3.mp4'
# specify frames per second
fps = 14.0
```

Final frames in a list:

```
frame_array = []
files = [f for f in os.listdir(pathIn) if isfile(join(pathIn, f))]
    files.sort(key=lambda f: int(re.sub('\D', '', f)))

    for i in range(len(files)):
        filename=pathIn + files[i]

        #read frames
        img = cv2.imread(filename)
        height, width, layers = img.shape
        size = (width,height)

        #inserting the frames into an image array
        frame_array.append(img)
```

Object Detection Video:

```
out = cv2.VideoWriter(pathOut,cv2.VideoWriter_fourcc(*'DIVX'), fps, size)

for i in range(len(frame_array)):
    # writing to a image array
    out.write(frame_array[i])

out.release()
```

Project explanation:

Objectives

The primary objective of the traffic management project is to create a comprehensive system that effectively monitors and manages traffic flow, reduces congestion, enhances safety, and provides data-driven insights for better urban planning. By utilizing IoT devices and cameras, the project aims to achieve the following specific objectives:

1. **Real-time Monitoring:** Implement a system that can continuously monitor traffic conditions in real-time across various intersections and road segments.
2. **Congestion Reduction:** Identify congestion-prone areas and implement strategies to alleviate traffic congestion and optimize the flow of vehicles.
3. **Safety Enhancement:** Utilize cameras and sensors to detect and respond to traffic violations, accidents, and other safety concerns promptly.
4. **Data Analysis:** Collect and analyze traffic data to generate valuable insights for making informed decisions regarding future infrastructure development and traffic management strategies.

IoT Device Setup:

The project involves the setup of various IoT devices, including High-Definition Cameras, Traffic Flow Sensors and Microcontrollers. These devices are strategically placed at critical points such as intersections and busy roads to capture real-time traffic data. The cameras are equipped with Night Vision Capabilities to ensure continuous monitoring regardless of lighting conditions. The traffic flow sensors are employed to detect vehicle movement and density accurately. Microcontrollers serve as the central processing units for data analysis and communication. The setup process involves careful placement and configuration of these devices, ensuring seamless data transmission to the central data management platform.

Platform Development:

The project's Central Data Management platform is designed to process, analyze, and visualize the data collected from the IoT devices. The architecture of the platform includes a centralized data processing system, a real-time monitoring dashboard, and a comprehensive database for storing historical traffic data and analytics. The platform employs a secure data sharing protocol for efficient transmission of data from the IoT devices. To ensure data security, the platform implements encryption protocols for secure data transmission and storage. Additionally, authentication mechanisms are integrated to regulate access to the platform, ensuring only authorized personnel can access the system.

Code Implementation:

The implementation of the project involves the use of various programming languages and frameworks. Python is utilized for IoT device programming and data processing, while a web-based framework is employed for the central data management platform. The codebase is organized into modules responsible for data collection, analysis, and visualization. Advanced image processing algorithms are developed to facilitate vehicle detection, license plate recognition, and traffic density estimation. These algorithms enable the system to accurately monitor traffic conditions and identify potential traffic violations and accidents promptly.

Conclusion:

The Traffic Management Project, driven by IoT devices and cameras, aims to revolutionize urban traffic management by providing real-time monitoring, congestion reduction, enhanced safety measures, and valuable data insights. The implementation of this system is expected to contribute significantly to the improvement of overall traffic conditions, leading to enhanced traffic flow, reduced congestion, and improved safety for all commuters. Moreover, the data-driven insights generated by the system will play a crucial role in making informed decisions for future urban development and infrastructure planning.

Team Members:

ARASU.C (810721106001 , arasu.c@care.ac.in)

SAMRUTH SRIRAM.D (810721106016, samruthsriram.d@care.ac.in)

HANISH.K.A (810721106007, hanish.ka@care.ac.in)

SIVAGANAPATHY.R (810721106018, sivaganapathy.r@care.ac.in)