PHASE 1: Problem Definition and Design Thinking

PROJECT: TRAFFIC MANAGEMENT SYSTEM

Problem Definition:

Urban areas face significant traffic congestion, leading to increased travel times, safety hazards, environmental pollution, and overall inefficiency. Existing traffic management systems often lack adaptability and integration of real-time data sources. Thus, there is a critical need to develop an IoT-based Traffic Management System that can effectively address these challenges by optimizing traffic flow, enhancing safety, reducing environmental impact, and improving overall urban mobility.

The problem at hand is the inefficient management of urban traffic, resulting in congestion, increased travel times, safety concerns, and environmental issues. This project aims to address this problem by developing an intelligent Traffic Management System (TMS) using IoT technologies. The key issues to be resolved include:

1) **Congestion and Delays:** Urban areas often experience traffic congestion during peak hours, leading to extended travel times and driver frustration.
2) **Safety Concerns:** Inefficient traffic management can contribute to accidents and collisions, endangering the lives of motorists and pedestrians.
3) **Environmental Impact:** Prolonged idling and stop-and-go traffic contribute to increased emissions, air pollution, and fuel consumption, negatively impacting the environment.
4) **Lack of Adaptive Control:** Existing traffic signal systems often operate on fixed schedules, leading to suboptimal traffic flow and inefficient resource utilization.
5) **Data Fragmentation:** There is a wealth of traffic-related data available from various sources, such as sensors, cameras, and GPS devices, but it is often underutilized or not integrated effectively.
6) **Ineffective Routing:** Traditional navigation systems do not adapt to real-time traffic conditions, leading to suboptimal routing decisions.
7) **Predictive Insights:** There is a need for predictive analytics to anticipate traffic patterns, accidents, and congestion events, allowing for proactive management.

8) **Communication and Coordination:** Effective communication and coordination between traffic signals, sensors, and vehicles are essential for a responsive traffic management system.
9) **User Experience:** The system should consider the user experience for both drivers and pedestrians by providing real-time updates and alternate routes.

The goal of this project is to develop a comprehensive IoT-based Traffic Management System that addresses these issues by seamlessly integrating data from various sources, implementing adaptive signal control, optimizing routing, and providing predictive insights to enhance traffic flow, safety, and environmental sustainability in urban areas.

Design Thinking:

Design thinking for an IoT-based Traffic Management System involves several key steps, including defining objectives, sensor design, and various other considerations. Here's a structured approach:

1. Objective Definition:

Understand the Problem: Begin by gaining a comprehensive understanding of the traffic management challenges in the chosen urban area. Identify key objectives such as reducing congestion, enhancing safety, and minimizing environmental impact.

Set Clear Objectives: Define specific, measurable, and achievable objectives for the IoT-based Traffic Management System. For example:

Reduce peak-hour congestion by 30%.

Decrease the number of accidents by 20%.

Lower carbon emissions by 15%.

2. Empathize:

User Needs Analysis: Conduct surveys, interviews, and observations to understand the needs and pain points of commuters, pedestrians, traffic authorities, and city planners.

3. Define:

Problem Definition: Clearly define the problem you aim to solve. For instance, "Develop an IoT-based Traffic Management System to optimize traffic flow, enhance road safety, and reduce environmental impact in urban areas."

4. Ideate:

Brainstorm Solutions: Organize brainstorming sessions with your team to generate ideas for the system. Consider technologies, approaches, and features that could address the defined problem.

5. Prototype:

Sensor Design: Develop sensor prototypes or specifications tailored to your system's objectives:

Traffic Flow Sensors: Design sensors that can detect vehicle presence, count, and speed to monitor traffic flow.

Safety Sensors: Create sensors capable of identifying safety hazards, such as sudden stops or erratic driving behaviour.

Environmental Sensors: Develop sensors to measure air quality and emissions.

Communication Sensors: Include communication modules to facilitate data transmission.

6. Test:

Sensor Testing: Evaluate the performance of the sensor prototypes in a controlled environment to ensure they meet accuracy and reliability requirements.

7. Iterate:

Refine Sensors: Based on the test results, iterate on the sensor design to improve accuracy, durability, and efficiency.

8. Implement:

Full System Development: Combine the refined sensors with the IoT platform and software components to create the complete Traffic Management System.

9. Monitor and Gather Data:

Deployment: Install the sensors and the system infrastructure in the selected urban area.

Data Collection: Continuously gather real-time data from the deployed sensors, including traffic flow, safety incidents, and environmental metrics.

10. Feedback Loop:

Data Analysis: Analyse the collected data to identify traffic patterns, congestionhotspots, safety issues, and environmental impact.

User Feedback: Engage with users and stakeholders to gather feedback on thesystem's effectiveness and user experience.

11. Iterate:

System Enhancement: Based on data analysis and user feedback, iterate on thesystem's algorithms and functionalities to further optimize traffic management.

12. Scale and Expand:

Scaling: Consider expanding the system to cover more areas or cities as itseffectiveness is demonstrated.

Partnerships: Explore partnerships with local government agencies and transportation companies to support the system's growth.

PHASE 2:INNOVATION

After thorough research and analysis, we arrived at an innovative solution to solve the above problem as detailed in phase 1 of our project.

- We will be using Raspberry Pi for the project. It is chosen for smart traffic management due to its affordability, compact size, low power consumption, GPIO pins for sensor integration, connectivity options, customizability with various operating systems, extensive software support, reliability, scalability, and open-source nature, making it a cost-effective and versatile solution for tasks like traffic monitoring, data collection, and signal control in traffic management systems.

SENSORS:

- **Camera Module:** The Raspberry Pi Camera captures real-time images and video footage of traffic flow. These visuals provide valuable insights into traffic conditions, allowing for real-time analysis.
- **Ultrasonic Sensors:** Ultrasonic sensors detect the presence of vehicles and measure the distance between vehicles. This data is crucial for understanding traffic density and congestion levels.
- **IR Sensors:** Infrared sensors are used for proximity detection and object recognition. They help in identifying obstacles or vehicles approaching specific areas, enabling efficient traffic management.

CONNECTIVITY:

Wi-Fi Dongle: If wired Ethernet connection is not feasible, a Wi-Fi dongle provides wireless connectivity, enabling Raspberry Pi to connect to the internet and transmit data to Azure cloud services.

AZURE CLOUD COMPONENTS:

Azure IoT Hub: Azure IoT Hub serves as the bridge between the Raspberry Pi-based system and the cloud. It enables secure communication, monitoring, and management of IoT devices, ensuring seamless data transmission.

Azure Blob Storage: Images and video footage captured by the Raspberry Pi Camera are stored in Azure Blob Storage. Storing this visual data in the cloud allows for historical analysis, pattern recognition, and evidence retrieval if needed.

Azure Stream Analytics: Stream Analytics processes real-time data streams from sensors and cameras. It performs instant data analysis, detecting traffic patterns, congestion points, and unusual events, facilitating timely responses.

Azure Functions: Azure Functions, based on serverless computing, are triggered by IoT data. They execute specific tasks in response to predefined events, enhancing automation and enabling customized actions based on real-time traffic conditions.

Azure Web Apps: Azure Web Apps create user-friendly dashboards and interfaces. These interfaces visualize traffic data, providing real-time updates to traffic authorities and the public. Users can access traffic information easily through web browsers or mobile devices.

Azure Cosmos DB (or other Database service): Azure Cosmos DB, a NoSQL database service, stores structured data from sensors and traffic events. It facilitates historical analysis, allowing traffic authorities to identify long-term trends, plan infrastructure changes, and optimize traffic management strategies.

Azure Virtual Machines (optional): Azure Virtual Machines can host custom applications or services related to traffic management. These applications can perform complex computations, machine learning algorithms, or AI-driven analytics to enhance traffic prediction and optimize traffic flow further.

PROTOCOL

HTTP:HTTP is an excellent choice for web based traffic management applications that require user interfaces and real time information delivery to end-users. They are suitable for traffic-related websites, mobile apps and dashboards.

PUBLIC PLATFORM

We are going to create a web-based or mobile dashboard to visualize real-time traffic data and system status.

Team members:

1. KAVIYA T V 2021504016
2. ANUSNEKHAA L N 2021504503
3. DAINTY CHRISTIANA J C R 2021504509
4. HARITHRA R 2021504520

PHASE 3:

PROBLEM:

To develop the python script on IoT devices as per the project requirement.

SOLUTION:

The Python script for IoT devices in the project utilizes the "IRremote" library to control traffic lights based on received IR remote commands, facilitating traffic management and signal changes.

SOURCE CODE:

```
#include <IRremote.h>
#define signal 11
int R1=2 , G1=3 , Y1=12 , R2=4 , G2=5 ,Y2=13 , R3=6 , G3=7 ,Y3=10  , R4=8 , G4=9 ;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  IrReceiver.begin(signal);
  for(int i=2 ; i<=9 ; i++){
      pinMode(i, OUTPUT);
  }

}

void Alert(int code){
  if(code==48){
      digitalWrite(R1, LOW);
      digitalWrite(G1, HIGH);
      digitalWrite(Y1, LOW);
      digitalWrite(R2, HIGH);
      digitalWrite(G2, LOW);
      digitalWrite(Y2, HIGH);
      digitalWrite(R3, HIGH);
      digitalWrite(G3, LOW);
      digitalWrite(Y3, LOW);
```

```
      digitalWrite(G1, LOW);
      digitalWrite(R4, HIGH);
      digitalWrite(G4, LOW);
      delay(3000);
    }
    else if(code==24){
      digitalWrite(R1, HIGH);
      digitalWrite(G1, LOW);
      digitalWrite(Y1, LOW);
      digitalWrite(R2, LOW);
      digitalWrite(G2, HIGH);
      digitalWrite(Y2, LOW);
      digitalWrite(R3, HIGH);
      digitalWrite(G3, LOW);
      digitalWrite(Y3, HIGH);
      digitalWrite(R4, HIGH);
      digitalWrite(G4, LOW);
      delay(3000);
    }
    else if(code==122){
      digitalWrite(R1, HIGH);
      digitalWrite(G1, LOW);
      digitalWrite(Y1, LOW);
      digitalWrite(R2, HIGH);
      digitalWrite(G2, LOW);
      digitalWrite(Y2, LOW);
      digitalWrite(R3, LOW);
      digitalWrite(G3, HIGH);
      digitalWrite(Y3, LOW);
      digitalWrite(R4, HIGH);
      digitalWrite(G4, LOW);
      delay(3000);
    }
    else if(code==16){
      digitalWrite(R1, HIGH);
      digitalWrite(G1, LOW);
      digitalWrite(Y1, HIGH);
      digitalWrite(R2, HIGH);
      digitalWrite(G2, LOW);
      digitalWrite(Y2, LOW);
      digitalWrite(R3, HIGH);
      digitalWrite(G3, LOW);
      digitalWrite(Y3, LOW);
      digitalWrite(R4, LOW);
      digitalWrite(G4, HIGH);
      delay(3000);
    }
}
int code=0;
void loop() {
  // put your main code here, to run repeatedly:
  if(IrReceiver.decode()){
    IrReceiver.resume();
    code=IrReceiver.decodedIRData.command;
```

```
  Alert(code);
}
digitalWrite(R1, LOW);
digitalWrite(G1, HIGH);
digitalWrite(Y1, LOW);
digitalWrite(R2, HIGH);
digitalWrite(G2, LOW);
digitalWrite(Y2, HIGH);
digitalWrite(R3, HIGH);
digitalWrite(G3, LOW);
digitalWrite(Y3, LOW);
digitalWrite(R4, HIGH);
digitalWrite(G4, LOW);
delay(3000);
if(IrReceiver.decode()){
  IrReceiver.resume();
  code=IrReceiver.decodedIRData.command;
  Alert(code);
}
digitalWrite(R1, HIGH);
digitalWrite(G1, LOW);
digitalWrite(Y1, LOW);
digitalWrite(R2, LOW);
digitalWrite(G2, HIGH);
digitalWrite(Y2, LOW);
digitalWrite(R3, HIGH);
digitalWrite(G3, LOW);
digitalWrite(Y3, HIGH);
digitalWrite(R4, HIGH);
digitalWrite(G4, LOW);
delay(3000);
if(IrReceiver.decode()){
  IrReceiver.resume();
  code=IrReceiver.decodedIRData.command;
  Alert(code);
}
digitalWrite(R1, HIGH);
digitalWrite(G1, LOW);
digitalWrite(Y1, LOW);
digitalWrite(R2, HIGH);
digitalWrite(G2, LOW);
digitalWrite(Y2, LOW);
digitalWrite(R3, LOW);
digitalWrite(G3, HIGH);
digitalWrite(Y3, LOW);
digitalWrite(R4, HIGH);
digitalWrite(G4, LOW);
delay(3000);
if(IrReceiver.decode()){
  IrReceiver.resume();
  code=IrReceiver.decodedIRData.command;
  Alert(code);
}
digitalWrite(R1, HIGH);
```

```
  digitalWrite(G1, LOW);
  digitalWrite(Y1, HIGH);
  digitalWrite(R2, HIGH);
  digitalWrite(G2, LOW);
  digitalWrite(Y2, LOW);
  digitalWrite(R3, HIGH);
  digitalWrite(G3, LOW);
  digitalWrite(Y3, LOW);
  digitalWrite(R4, LOW);
  digitalWrite(G4, HIGH);
  delay(3000);
}
```

LIBRARIES:

IR Library: This library is commonly used with Arduino to work with IR control signals. It provides functions to receive and decode IR signals from various remote controls. With this library, you can determine the IR code associated with a button press and use that code to trigger actions in your Arduino project.
You can install this library through the Arduino IDE's Library Manager, making it available for use in your Arduino sketches.

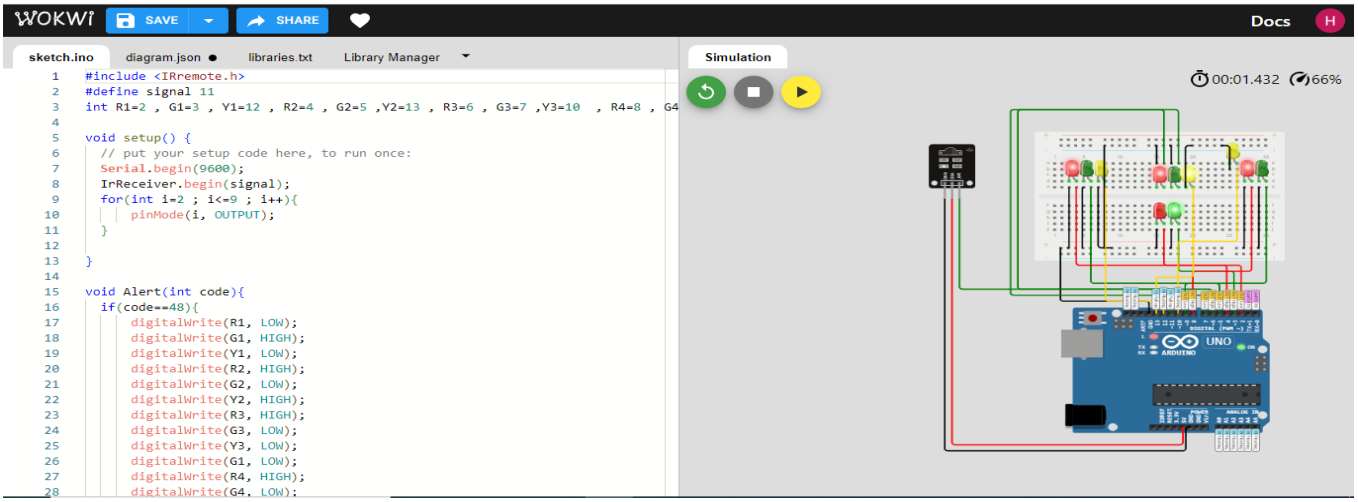
In addition to the "IR" library, your code doesn't include any other external libraries. It mainly relies on standard Arduino functions and the built-in libraries for GPIO control.

To use the "IR" library in your project, ensure that it is correctly installed in your Arduino IDE, and your code should work as intended for controlling the traffic light system based on IR remote commands**.**

LINK:

https://wokwi.com/projects/379578847110009857

SIMULATION:



Team members:

1. KAVIYA T V 2021504016
2. ANUSNEKHAA L N 2021504503
3. DAINTY CHRISTIANA J C R 2021504509
4. HARITHRA R 2021504520