public transportation optimization

By using IOT (Internet Of Things) technology

Introduction

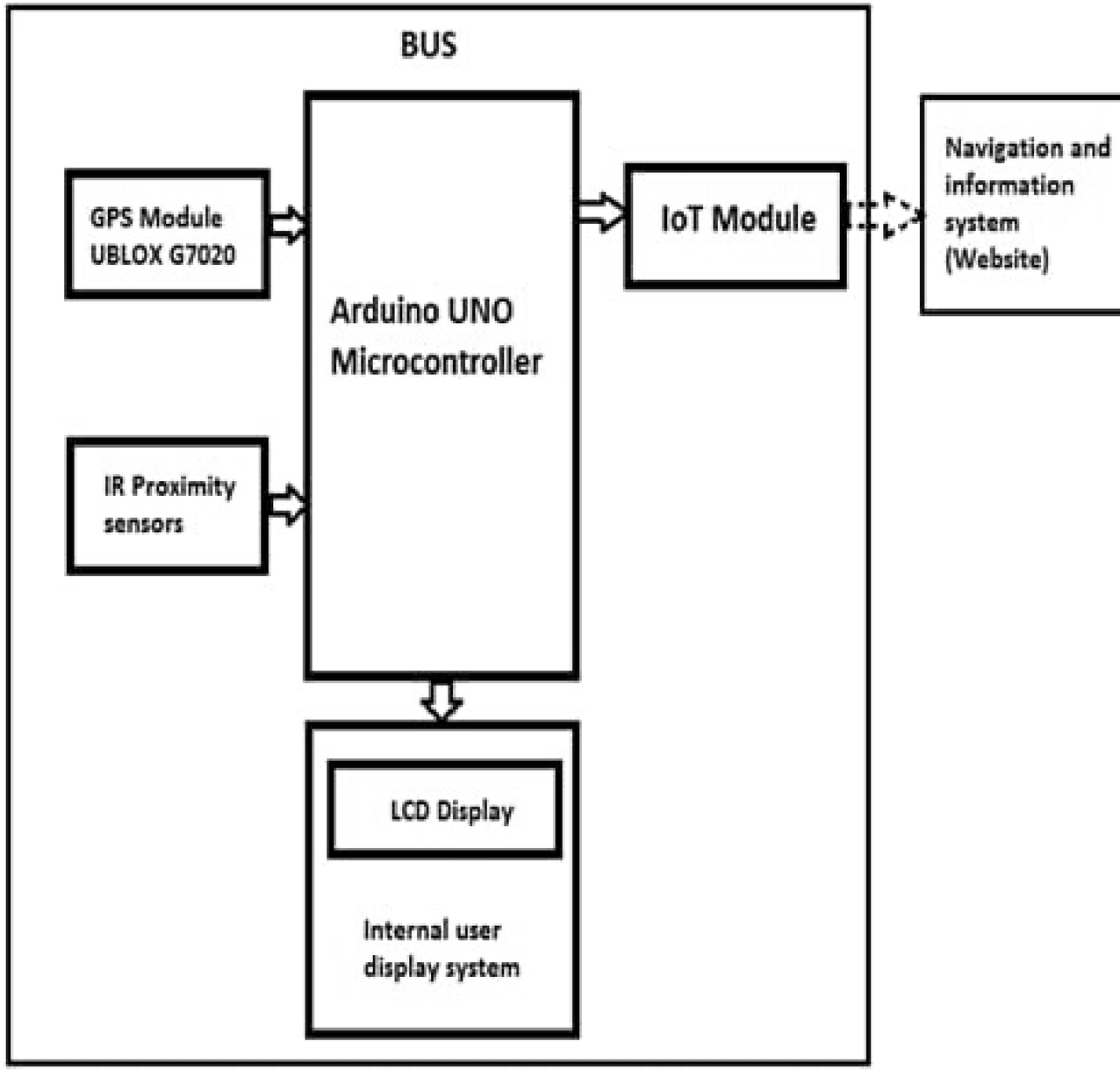
In this project, we are going to implement the IOT based public transportation optimization for disabled person

Main Application is,
Boarding and alighting vehicles

- Drivers not waiting until passengers are seated before moving
- No one to provide assistance in getting on/off the vehicles.
- Bus stops located at inconvenient places, often with no form of shelter

Description

The aim is to make public transport a convenient and equitable mode of transportation for everyone, regardless of their physical or cognitive abilities. This optimization helps promote independence and social inclusion for people with disabilities while fostering a more inclusive society



working

1St makes a public transportation outlook like a train or bus in our case we use the bus Outlook.

Project works normally we can control frd,brd,left, right direction.

when the transportation vehicle reach the stoping area the door will open automatically after the passanger will scan they travel card in a RFID scanner.

it will show passenger validation I'd say the door will close after all the passengers come inside the vehicle.

the door camera will scan the passenger who is disabled or not. so that they gave special area to the passenger.

an display and voice assistant will give instructions on what stop will come next.

Required components

ESP32 CAM MODULE

ESP32 DEV BOARD

SG 90 SERVO

GPS MODULE

BO MOTOR

12V 2500MAH LI-ION BATTERY

TRANSPORTATION OUTLOOK SETUP

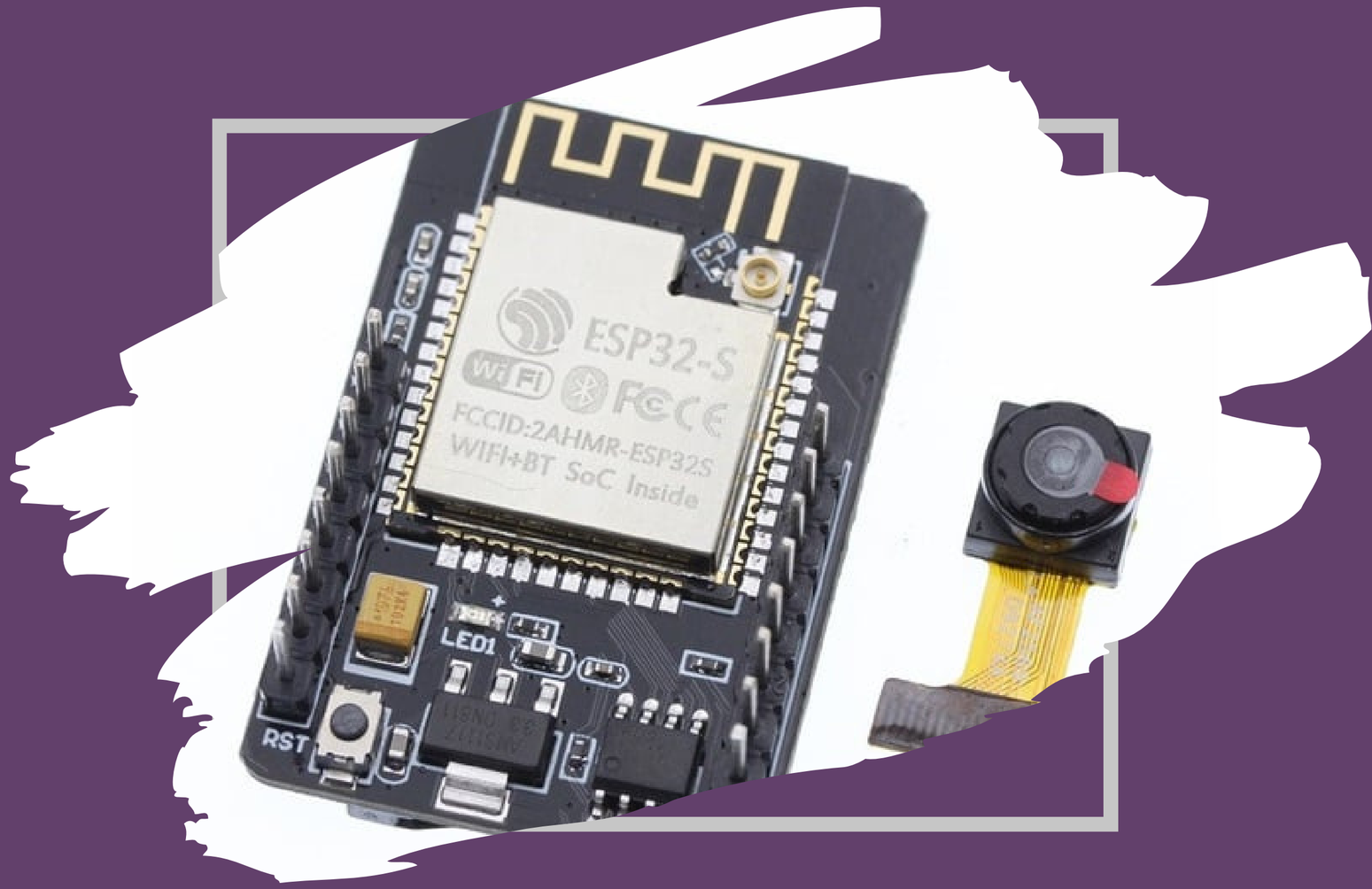
ACCIDENT SENSOR SWITCH

REGULATOR MODULE

L298N MOTOR DRIVER

Esp 32 cam module

The ESP32-CAM is a small size, low power consumption camera module based on ESP32. It comes with an OV2640 camera and provides onboard TF card slot. The ESP32-CAM can be widely used in intelligent IoT applications such as wireless video monitoring, WiFi image upload



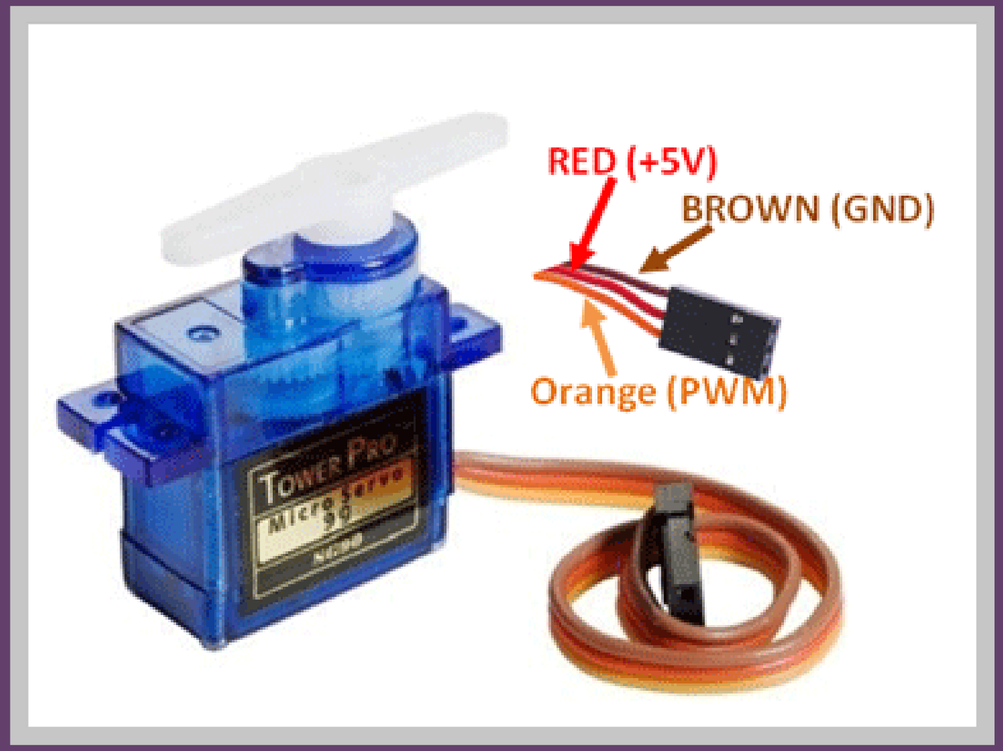
coding

```
import network
import urequests as requests
import time
from machine import Pin
import network
import machine
import camera
# Set up your Wi-Fi credentials
WIFI_SSID = "YourWiFiSSID"
WIFI_PASSWORD = "YourWiFiPassword"
# Set up endpoint to send captured images for analysis
SERVER_URL = "http://your_server_url"
# Initialize the ESP32-CAM
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(WIFI_SSID, WIFI_PASSWORD)
while not wlan.isconnected():
    time.sleep(1)
# Initialize the camera
camera.init()
# Create a pin for triggering image capture
button = Pin(0, Pin.IN)
```

```
# Main loop
while True:
    # Capture an image when the button is pressed
    if button.value() == 0:
        img = camera.capture()

        # Send the captured image to the server for
analysis
        response = requests.post(SERVER_URL, data=img,
headers={'Content-Type': 'image/jpeg'})

        # Process the server's response to determine bus
arrival time
        bus_arrival_time = response.text  # You need to
adapt this based on your server's response format

        print(f"Bus Arrival Time: {bus_arrival_time}")



.time.sleep(1)
```

Sg 90 servo motor

Micro Servo Motor SG90 is a tiny and lightweight server motor with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller. You can use any servo code, hardware or library to control these servos.

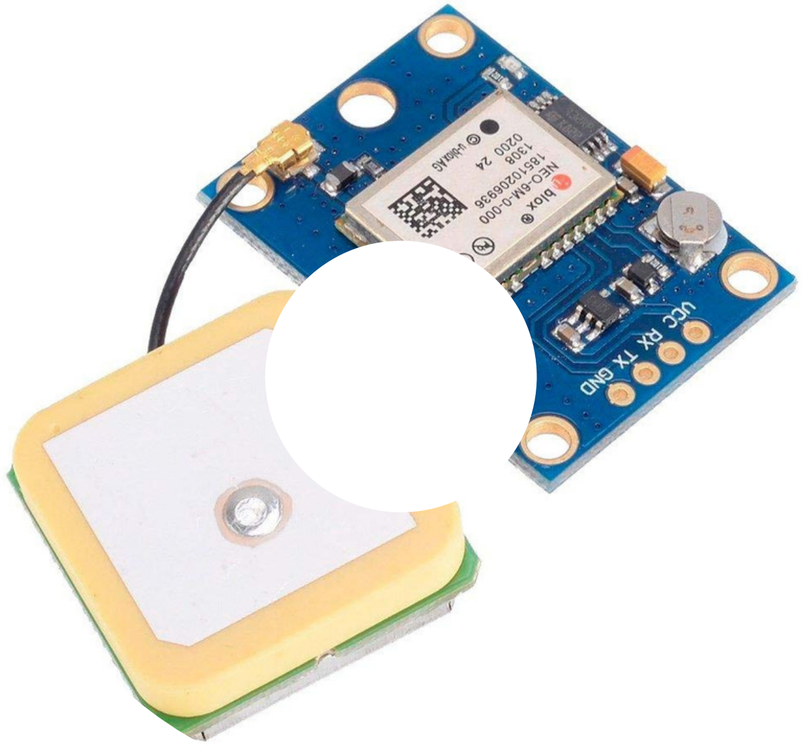


coding

```
#include <Servo.h>

Servo myservo;  // Create a servo object

void setup() {
myservo.attach(9);  // Attach the servo to pin 9
}

void loop() {
int pos;

// Move the servo from 0 to 180 degrees
for (pos = 0; pos <= 180; pos += 1) {
myservo.write(pos);  // Set the servo position
delay(15);          // Wait for the servo to reach the new position
}
delay(1000);          // Pause at 180 degrees

// Move the servo from 180 to 0 degrees
for (pos = 180; pos >= 0; pos -= 1) {
myservo.write(pos);  // Set the servo position
delay(15);          // Wait for the servo to reach the new position
}
delay(1000);          // Pause at 0 degrees
}
```

GPS module

The GPS module is a wireless chip module combined on the mainboard of a mobile phone or machine. It can communicate with the global satellite positioning system in the United States. It can locate and navigate according to the condition of a wireless network signal

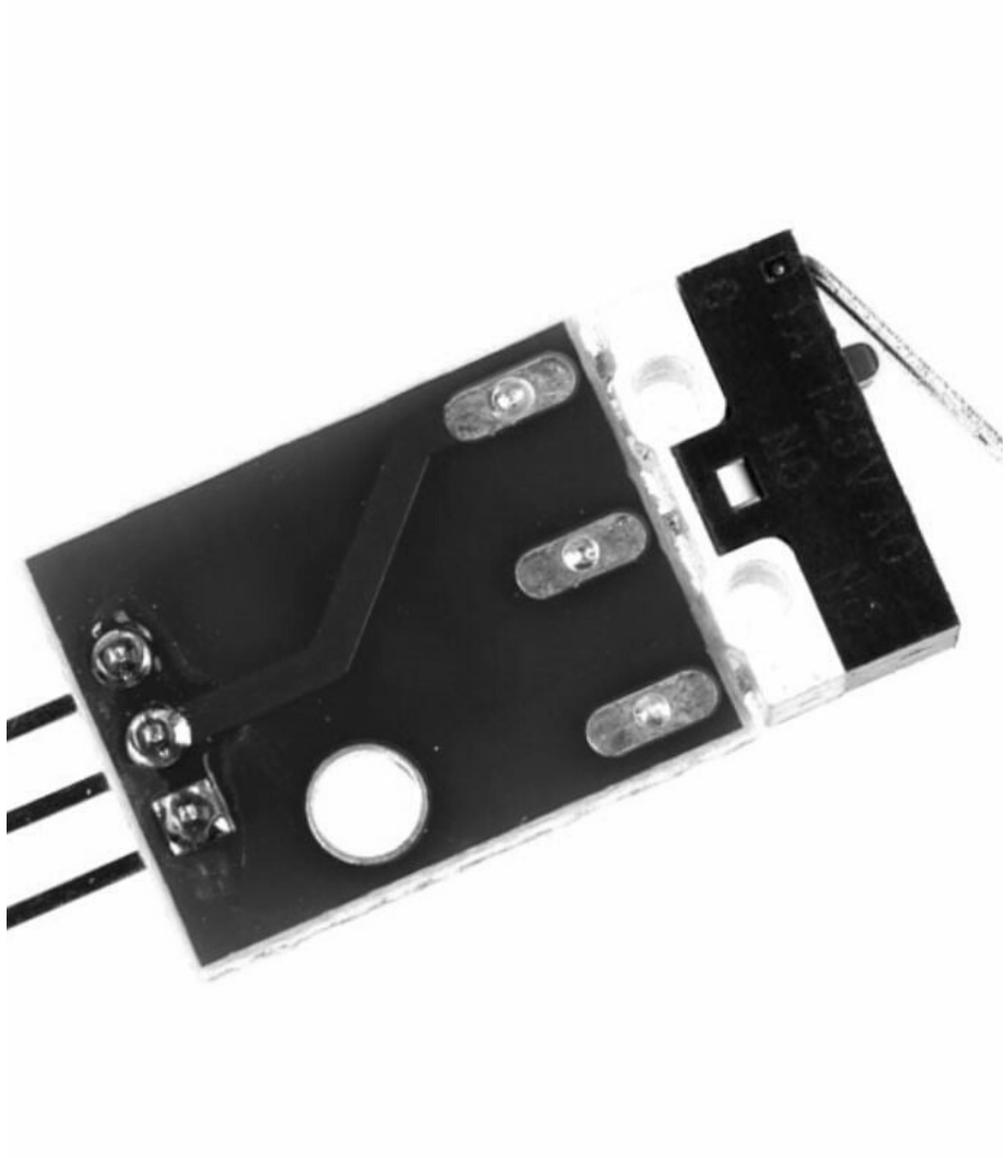


coding:

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
static const int RXPin = 4, TXPin = 3;  // Define your RX and TX pins
static const uint32_t GPSBaud = 9600;
SoftwareSerial ss(RXPin, TXPin);
TinyGPSPlus gps;
void setup() {
Serial.begin(115200);  // Serial monitor
ss.begin(GPSBaud);    // GPS module
}
void loop() {
while (ss.available() > 0) {
if (gps.encode(ss.read())) {
if (gps.location.isValid()) {
double latitude = gps.location.lat();
double longitude = gps.location.lng();
// Use latitude and longitude for route optimization and navigation
}
}
```

Accident switch sensor

detect an object in the path of a moving vehicle so the human operator, or the vehicle's automated system, can take action to avoid a collision.



coding

```
const int sensorPin = A0;  // Analog pin for reading sensor data
const int threshold = 100; // Set your threshold value

void setup() {
pinMode(sensorPin, INPUT);
Serial.begin(9600);
}

void loop() {
int sensorValue = analogRead(sensorPin);

if (sensorValue > threshold) {
// Collision or impact detected
Serial.println("Collision detected!");
// Trigger actions, such as airbag deployment, alarms, etc.
}

delay(100); // Adjust the delay as needed
}
```

problem solution:

```
from pulp import LpProblem, LpMinimize, LpVariable
# Create a Linear Programming problem
problem = LpProblem("Transport_Optimization", LpMinimize)
# Define decision variables (number of buses on each route)
route_A = LpVariable("Route_A_Buses", lowBound=0,
cat='Integer')
route_B = LpVariable("Route_B_Buses", lowBound=0,
cat='Integer')
route_C = LpVariable("Route_C_Buses", lowBound=0,
cat='Integer')
# Define the objective function (minimize total cost)
problem += 1000 * route_A + 1200 * route_B + 1500 * route_C,
"Total_Cost"
```

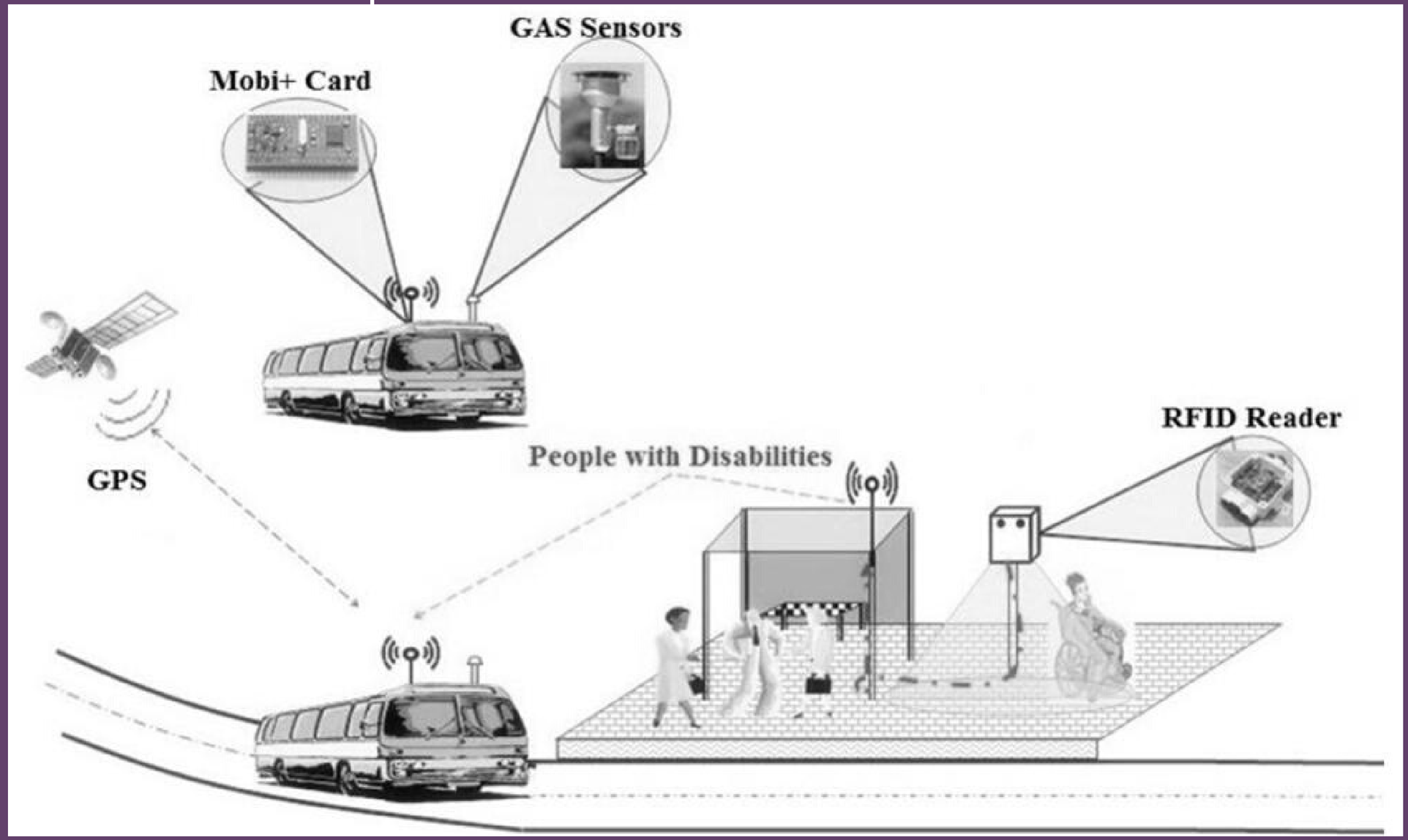
```
# Define constraints (e.g., capacity constraints)
problem += route_A <= 30, "Route_A_Capacity"
problem += route_B <= 40, "Route_B_Capacity"
problem += route_C <= 20, "Route_C_Capacity"
problem += route_A + route_B + route_C >= 60, "Total_Capacity"

# Solve the problem
problem.solve()

# Print the results
print("Route A buses:", route_A.varValue)
print("Route B buses:", route_B.varValue)
print("Route C buses:", route_C.varValue)
print("Total cost: $", problem.objective.value())
```

Program for optimization

```
import networkx as nx
# Create a graph representing the transportation network
G = nx.DiGraph()
# Add nodes representing stops or locations
G.add_nodes_from(['A', 'B', 'C', 'D', 'E', 'F', 'G'])
# Add edges representing connections between stops with travel times
G.add_edge('A', 'B', weight=5)
G.add_edge('A', 'C', weight=8)
G.add_edge('B', 'D', weight=4)
G.add_edge('C', 'D', weight=6)
G.add_edge('D', 'E', weight=3)
G.add_edge('D', 'F', weight=7)
G.add_edge('E', 'G', weight=5)
G.add_edge('F', 'G', weight=6)
# Find the shortest path from 'A' to 'G'
shortest_path = nx.shortest_path(G, source='A', target='G', weight='weight')
print("Shortest Path:", shortest_path)
```

IBM Cloud

IBM Cloud Paks are software products for hybrid clouds that enable you to develop apps once and deploy them anywhere.
Virtual Private Cloud (VPC) is available as a public cloud service that lets you establish your own private cloud-like computing environment on shared public cloud infrastructure.


GAS Sensors
Mobi+ Card
GPS
People with Disabilities
RFID Reader

Thankyou!!