Automated Vehicle Control Systems

The service layer the user interacts with is made up of Vehicles and a distributed internet framework. The network is made up of node controllers and WiFi routers. The transport layer is base on UDP. The roads are represented by an edge centric graph and uses Djikstra's Algorithm for shortest path. The vehicle status is held in the vehicle and on the network. The port numbers serve as keys. Pathing is controlled by the network. The vehicle handles local control and safety.

Components:

PC and Packet Sender: Right now the users requests to the network are simulated by creating a UDP message and sending it to the network. Debugging vehicle operations also using message to simulate network commands.
Vehicle: Each vehicle has a NodeMCU ESP32S microcontroller, a 7.4v battery, 2 dc motors, a qrtcc 8 sensor IR reader, and an L298N motor controller and voltage regulator. The vehicle is for now essentially a line follower that makes decisions at intersections.
Node: The node is a NodeMCU ESP32S microcontroller. It contains a class for vehicles and for the city. It holds the edges and performs all controls for vehicles in the local area.

Protocol:

The communication protocol is UDP. Each packet is a string with the following format:

```
String temp = "Type:  ";
temp = temp + TYPE
      " Name: " + vName +
      " Loc: " + location +
      " Dest: " + destination +
      " NextLoc: " + nextLocation +
      " Addr: " + destinationAddress +
      " Pass: " + passengers +
      " CObj: " + currentObjective +
      " Spd: " + baseSpeed +
      " DStatus: " + driveStatus +
      " NxtVrtType: " + nextVertType;
```

Each string is followed by an integer value. There are other formats used for debugging but this is the only public facing packet. Each function pulls different values from the message based on the TYPE variable that denotes the type of message.

Graph Structure:

The map here is represented by a graph of Edges. Each edge uses the following struct:

```
struct edge {
        int eName;
        int destType;
        int weight;
        int index;
        int leftTurn;
        int rightTurn;
        int straight;
```

Malloc and New were problematic with the Arduino IDE and the memory and hardware limitations so pointers were simulated by an array and using values as indexes. The vehicle passes through vertices. It never stops there or stays there so vertices are reduced to turn types.

When a request is made by the server the destination is changed. The shortest path is determined by a variation of Djikstra's shortest path algorithm. Once the path is written the vehicle will follow it to the end. Constant updates to the shortest path would be fairly easy to set up.

Line Following Algorithm:

Correction = (Position - reference) * Kposition + (Position - Old position) * Kdelta

Limitations: Currently the system is not very consistent. Small wrinkles in the paper from being rolled up last night are causing problems. The line following algorithm and hardware is flakey in general. I look forward to replacing this system with a video reading program that analyzes video instead of IR which is affected by ambient light and wrinkles.