TRUCK PLATOONING SYSTEM

Team: SVSH

Supadma Kadabi

Vibhashree Hippargi

Sachin Kumar

Hamida Aliveya

Git: https://github.com/VibhaHippargi/DPSTruckPlatooning/tree/final

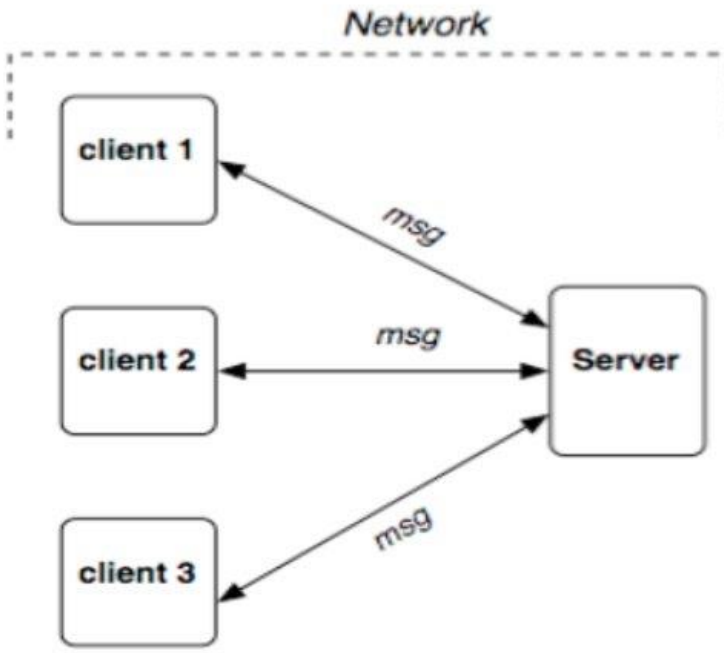
What is Platooning?

Platooning is an example of a value-added service that calls for architectural changes from an existing automobile to a future smart car, a change from a feature phone to a fixed-function embedded system, and a change to a single software framework that can connect different services to the electronic units of the vehicles inside a platoon, especially for heavy-duty trucks, results from lower air drag.

In the project, there is a leader truck which contain human driver and the other platooned trucks that are automated. They donot contain the human driver. They follow the leader truck and execute the instructions provided by it.

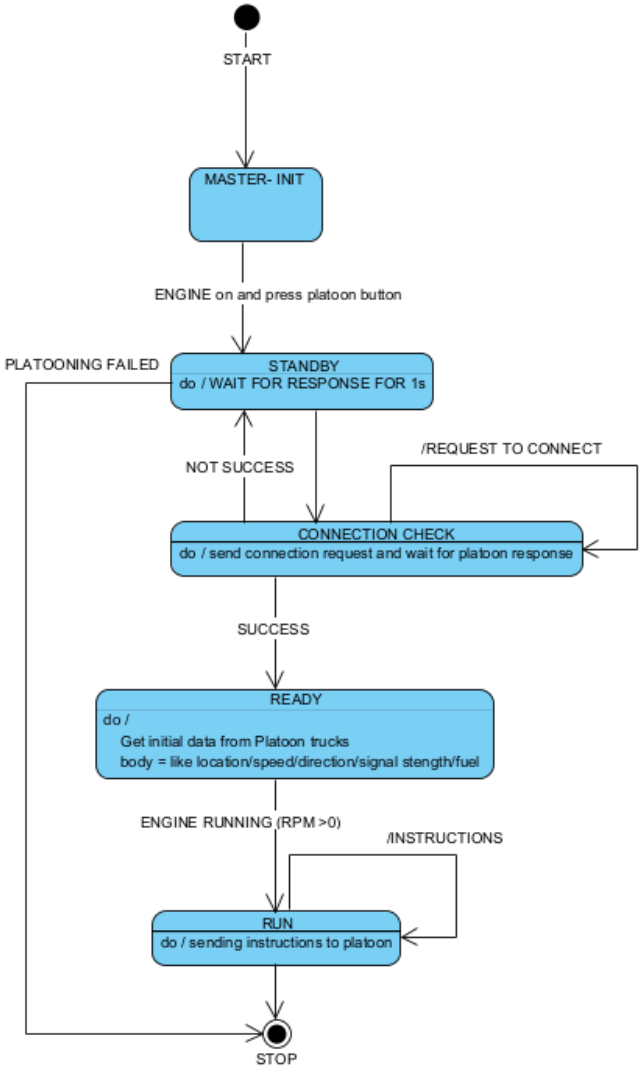
CLIENT – SERVER ARCH.

Distributed systems which communicate with each other over the network are usually addressed as Client-Server architecture model. The below figure shows the pictorial representation of a server that is connected to 3 clients through a network. They communicate with each other by exchanging messages (msg in the diagram).

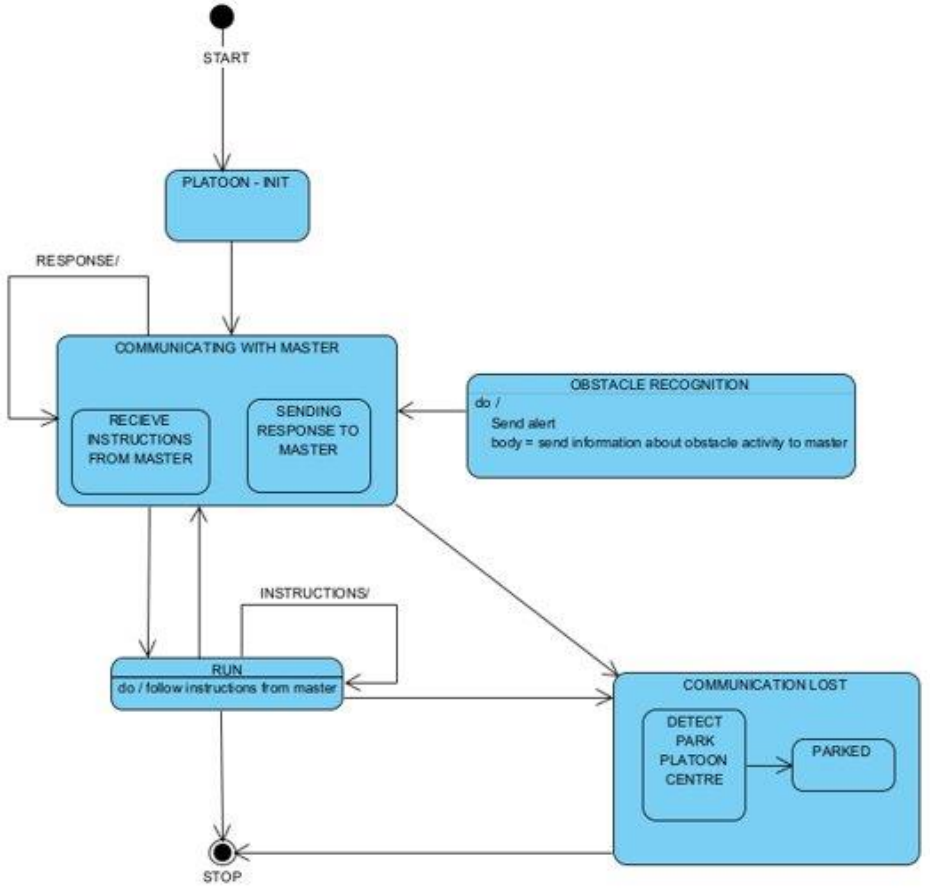


State Diagram

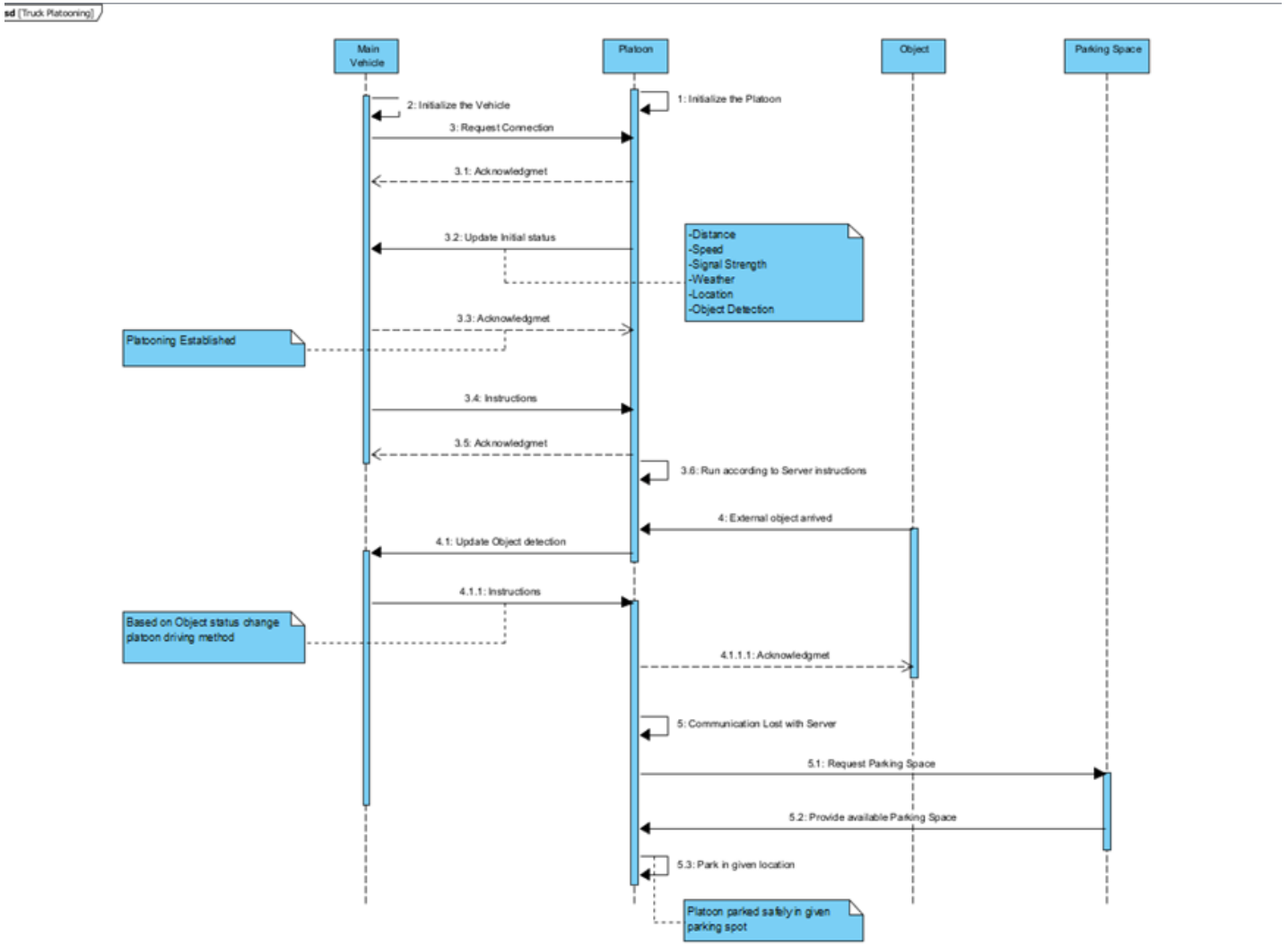
LEADER TRUCK



PLATOONED TRUCKS

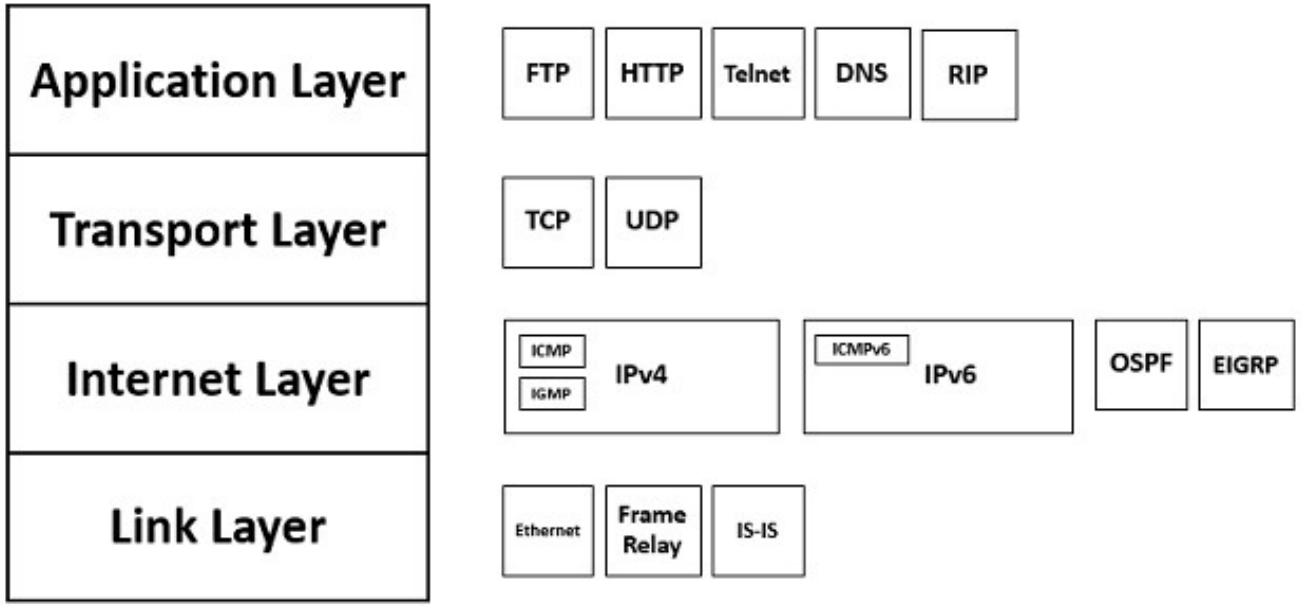


Sequence Diagram



TCP Model

A protocol is a set of rules that is to be followed for proper communication between the devices on a network. It ensures that communication is carried out in a consistent and predictable way. TCP/IP model is a conceptual framework that defines how the devices send and receive messages with each other over a network.

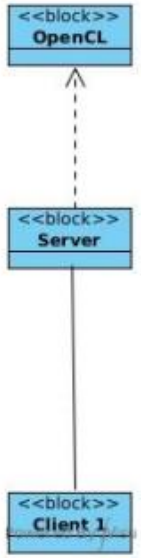


GPU Implementation

The GPU (Graphical Processing Unit) is another processor unit with the Host CPU (Central Processing Unit). It plays an important role where the tasks must be performed in parallel e.g. 3D rendering.

Available Framework: CUDA, OpenCL, One API,HIP etc

OpenCL provides an approach where the large number of GPUs are targeted from different vendors. OpenCL (Open Computing Language) is standard for cross-platform and parallel processing. It is a low level api for heterogeneous programming. In OpenCL, the kernels are defined which are the functions which execute on the GPU threads.



Topology:

The topology used is based on the Server and Client, where the client platoons connect with the server, which is the main platoon and then server uses the kernel in order to process the data in parallel in the GPU.

- Server socket creation
- Client socket creation
- Employ OpenCL library
- Checking GPU Platform
- Kernel to compare distance and safety distance
- Buffer creation and data writing
- Adding arguments to kernel and execution

Code Structure

1. Server:

The Server class uses the C++ socket library to create the socket and binds the server to the port 8080. It has a *run* function which waits for the clients to connect and then gets the data of vehicles.

```
class Server {
public:
    Server() {…
    ~Server() {…

    void run() {…

private:
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt;
    int addrlen;
    char buffer[1024] = { 0 };
    char* hello;
    ClCode *clCodeObj;
};
```

***Server* Class Structure**

2. Client

The Client class also uses C++ socket library to create the socket and then use a *run* function which connects with the socket server and then sends the data to the server and also, the server sends the data back to client in order to recognize that the data is received successfully and send the next data.

```
class Client {
public:
    Client() {…
    ~Client() {…
    void run() {…
private:
    int sock, valread, client_fd;
    struct sockaddr_in serv_addr;
    char* hello;
    char buffer[1024] = { 0 };

};
```

***Client* Class Structure**

Code Structure

3.ClCode:

The ClCode class uses the OpenCL library to create the kernel, which is a function used to execute the logic on the GPU threads.

```
class ClCode {
    public:
    ClCode();
    ~ClCode();
    void distanceCompare(std::size_t numberOfVehicles,
    int vehicleDistances[], int safetyDistances[], bool status[]);
    private:
    std::vector<cl::Platform> all_platforms;
    cl::Context *context;
    cl::Device default_device;
    cl::Program *program;
};
```

***ClCode* Class Structure**

4. Kernel

The above code is used as a kernel to execute the data on the GPU threads. Once, the distance data of all the vehicles is received, then it calls the ClCode instance *distanceCompare* function which sends the data to the GPU kernel in order to perform the logic where it needs to return whether the vehicle should accelerate or decelerate in order to maintain the safety distance.

```
"   void kernel distance_safety(global const int* vehicleDistance, global const int* safetyDistance, global bool* status){
"         if(vehicleDistance[get_global_id(0)] > safetyDistance[get_global_id(0)]) {                    "
"             status[get_global_id(0)] = true;                                                  "
"         }                                                                                     "
"         else {                                                                                "
"             status[get_global_id(0)] = false;                                                 "
"         }                                                                                     "
"   }                                                                                           ";
```

Kernel to compare distance and safety distance

Code Structure

5.Buffer Creation and Data Writing

The sequence of the execution of the code follows a certain standard in which firstly, the buffers on the device are created, which are used to communicate between the host (CPU) and the device (GPU). After this, the data from the host is written into the buffer data type variables.

```
// create buffers on the device
cl::Buffer buffer_vehicleDistances(*context,CL_MEM_READ_WRITE,sizeof(int)*numberOfVehicles);
cl::Buffer buffer_safetyDistances(*context,CL_MEM_READ_WRITE,sizeof(int)*numberOfVehicles);
cl::Buffer buffer_status(*context,CL_MEM_READ_WRITE,sizeof(bool)*numberOfVehicles);


//create queue to which we will push commands for the device.
cl::CommandQueue queue(*context,default_device);

//write arrays A and B to the device
queue.enqueueWriteBuffer(buffer_vehicleDistances,CL_TRUE,0,sizeof(int)*numberOfVehicles,vehicleDistances);
queue.enqueueWriteBuffer(buffer_safetyDistances,CL_TRUE,0,sizeof(int)*numberOfVehicles,safetyDistances);
```

Buffer creation and data writing

6. Adding arguments to kernel and execution

The arguments are set on the kernels and then enqueue the kernel in order to execute the kernel in the GPU threads. After the execution, the buffer result data is read back to the host data type variable in order to further process the data.

```
//run the kernel
cl::Kernel kernel_add=cl::Kernel(*program,"distance_safety");
kernel_add.setArg(0,buffer_vehicleDistances);
kernel_add.setArg(1,buffer_safetyDistances);
kernel_add.setArg(2,buffer_status);
queue.enqueueNDRangeKernel(kernel_add,cl::NullRange,cl::NDRange(numberOfVehicles),cl::NullRange);
queue.finish();

printf("queue finish\n");
//read result C from the device to array C
queue.enqueueReadBuffer(buffer_status,CL_TRUE,0,sizeof(bool)*numberOfVehicles,status);
```

Adding arguments to kernel and execution

Code Execution

The execution of the programs server and client is shown in the below figures, which use the command line arguments in order to send the data of the vehicles from the client to the server and then from the server to the GPU device. The client socket connects with the server and sends the distance values to the server. The server sends back the message in order to recognize the data has been received.

```
Number of vehicles: 5
ok
Enter distance for vehicle 0: 40
ok
Enter distance for vehicle 1: 60
ok
Enter distance for vehicle 2: 70
ok
Enter distance for vehicle 3: 40
ok
Enter distance for vehicle 4: 90
Vehicle 0 should Accelerate.
Vehicle 1 should Decelerate.
Vehicle 2 should Decelerate.
Vehicle 3 should Accelerate.
Vehicle 4 should Decelerate.
```

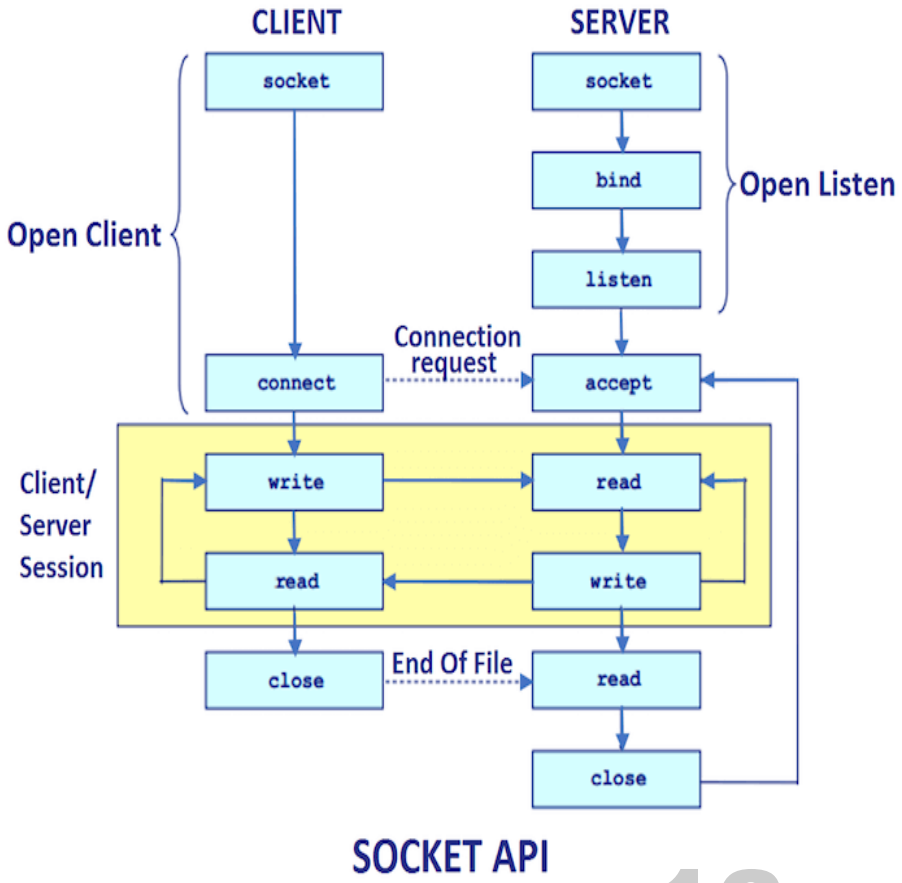
Client CLI

```
size of platforms 1
Using platform: NVIDIA CUDA
Using device: NVIDIA GeForce RTX 3060 Laptop GPU
server is listening
number of vehicles 5
vehicle value received
vehicle value received
vehicle value received
vehicle value received
vehicle value received
queue finish
```

Server CLI

JAVA implementation:

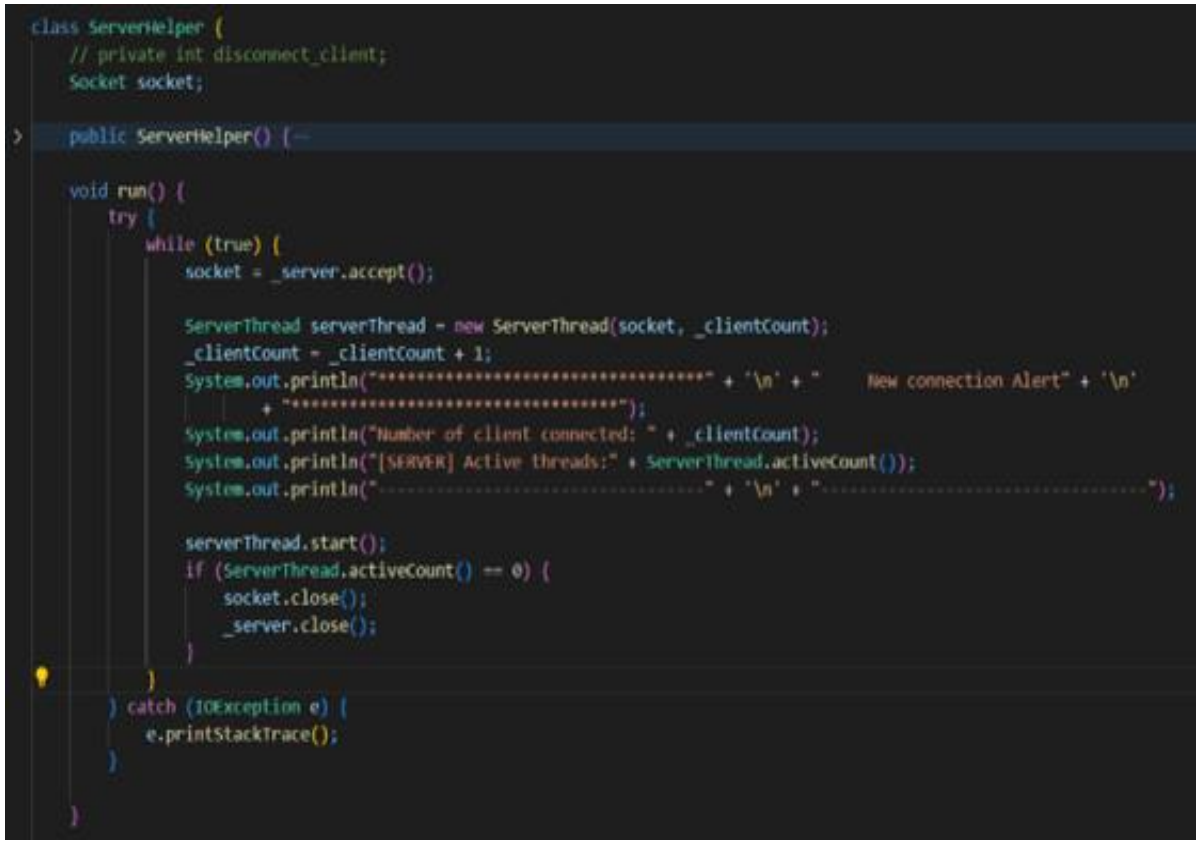
- Socket Programming is used
- The java programming language provides advanced classes and libraries which are user-friendly and less prone to attacks.
- Socket programming in java is connection based. It has classes like ServerSocket and Socket which contain in-built methods for most of the required functionalities.
- This programming language is more advanced. The Socket class of Java has various combinations of constructors to use it with Port, IP address, and number of backlog needed which can be used as the situation demands.
- bind, listen and accept are internally handled by this class



Details on Implementation

Implementation of Server:

1. In this system multiple classes are used to perform the different functionalities. The server contains different classes like *Server, ServerThread, ServerHelper* which combined perform all the basic functionalities of the Main Vehicle. The *ServerHelper* class mainly controls the client connection and disconnections. It also displays required comments for understanding the system status.
2. The *serverThread* class contributes in interacting with all the computational classes for instructing the client with next steps to be taken. It is the main interface between the platoons and the Main Vehicle.



Details on Implementation

3. The *monitorPlatoonData* is used to monitor each variable updated by the platoon viz. *monitor_distance(), monitor_signal_strength(), monitor_speed(), Object_detection_status(), Weather_monitoring().*

These methods further interact with Velocity_equations class for calculating the values at the lowest level possible. The function Weather_monitoring() helps the server to understand the environment and road conditions, using this method it updates the Platoons about the weather and the caution to be taken while driving.

4. The *Veloctiy_equations* uses the basic displacement equations to calculate the values of acceleration and time using simple variables like *accerleration_fin, time, ini_vel, fin_vel, distance* etc. It also converts the measurement from m/sec to km/hr before and after calculations for better understanding and application purposes.

```java
public class Velocity_equations {

    // rounding off the final out come to 2 decimal places
    private static final DecimalFormat dec = new DecimalFormat(pattern: "0.00");
    public double acceleration;
    public double acceleration_fin;
    public double time;

    public String vel_calculation(int ini_vel, int fin_vel, double distance) {
        // convert velocity to m/s, so we divide the given kmph by 3.6 to m/s
        // v^2 -u^2 = 2as, V-> Final Velocity, u -> initial Velocity a -> acceleration,
        // s -> distance
        // Standard distance is 40m, Platoon is at 50m, then it has to cover 10 m with
        // given acceratation
        double ini_vel_m = (ini_vel / 3.6);
        double fin_vel_m = (fin_vel / 3.6);
        acceleration = ((Math.pow(fin_vel_m, b: 2)) - (Math.pow(ini_vel_m, b: 2))) / (2 * distance);
        // convert accelration back to kmph^2
        acceleration_fin = acceleration * 3.6;

        return (dec.format(acceleration_fin));
    }

    public String time_calculation(int ini_vel, int fin_vel) {
        // calculate the time for which the acceleration needs to be maintained
        // v = u + at, a-> calculated from previous function, t -> time
        double ini_vel_m = (ini_vel / 3.6);
        double fin_vel_m = (fin_vel / 3.6);
        time = (fin_vel_m - ini_vel_m) / (acceleration);

        return (dec.format(time));
    }
}
```

Details on Implementation

There are individual classes defined in the system for Main vehicle and Platoon which handle all the variables that are required for monitoring the system. These classes are defined as **serializable** because they are used to transfer the values between the classes serially using buffers.

```
public class leader implements Serializable {

    /*...
    private double distance;
    private int speed, signal_strength;
    private String weather;
    private Location location;
```

Class Leader

```
public class platoon implements Serializable {

    private double distance;
    private int speed, signal_strength;
    private Location location;
    private boolean quit;
    private int Object_detected_inmtrs;
    private boolean object_detection;
```

Class Platoon

Details on Implementation

Implementation of Client:

The client class in this system describes the Platoons, it initiates all the platoon variables and sends them to the Main vehicle on regular intervals. Here it is realized by clicking the enter button for easy usage. In this system, the standard IP address “127.0.0.1” and standard port “9090” is used for connection.

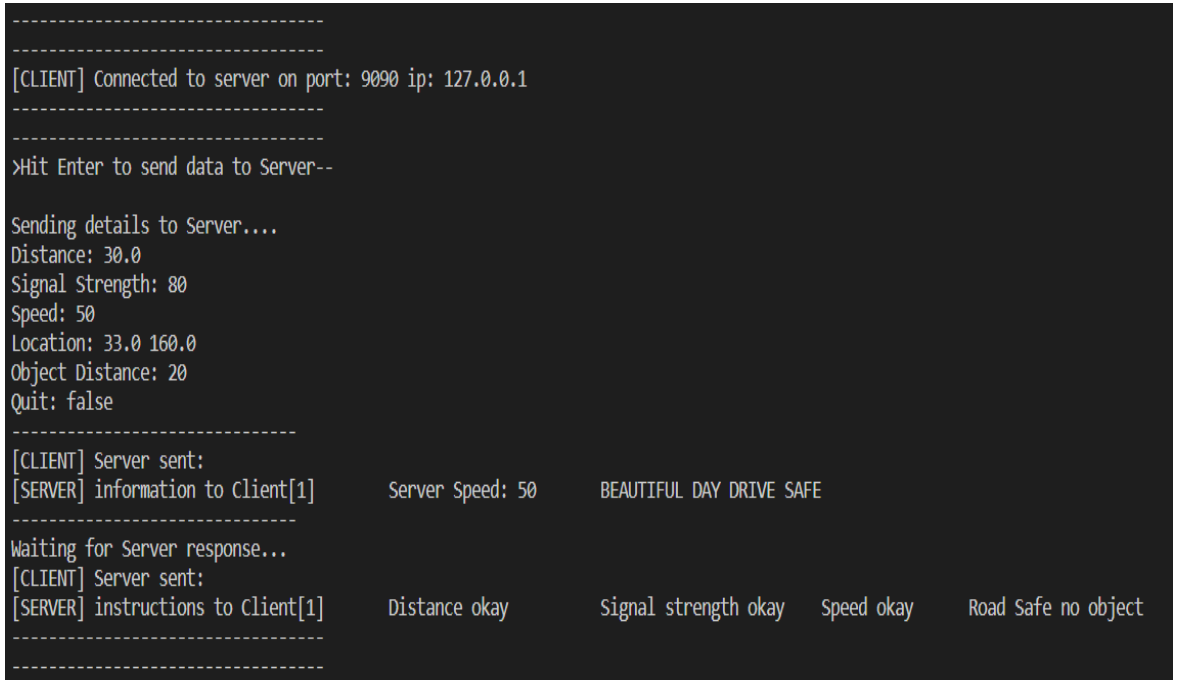
Two functionalities related to Platoons are:

- Object Detection : This is handled by variables *Object_detected_inmtrs, object_detection* and the method *monitor_object_distance().* The platoon constantly verifies if the object that is an intruder car is approaching itself and updates the Main vehicle when it is less than 10m away from it.

- Signal Strength: There are two types of signal strength information in this system, one where the Main vehicle verifies if platoon is still able to connect to it and the other is platoon checking for its ability to connect with Main vehicle. When the signal strength is less than platoon expected strength, the platoon checks for nearby parking locations and updates the parking location to the server.

Results and Discussions

1. Normal Conditions: When the platoon is following the Main Vehicle and the values are maintained as per Main Vehicle, it acknowledges the platoon to continue in the same state



Output Normal conditions

2. Platoon slower than Leader : When the platoon is slower than the Leader vehicle, then the leader updates the platoon to increase the speed by giving it the exact values for acceleration and time

Platoon slower than Leader

Results and Discussions

3. Platoon faster than Leader : When the platoon is traveling at a higher speed and is approaching the next vehicle, the Main vehicle instructs it to slow down.

```
[CLIENT] Connected to server on port: 9090 ip: 127.0.0.1
---------------------------------
---------------------------------
>Hit Enter to send data to Server--

Sending details to Server....
Distance: 10.0
Signal Strength: 50
Speed: 60
Location: 68.0 178.0
Object Distance: 20
Quit: false
------------------------------
[CLIENT] Server sent:
[SERVER] information to Client[1]       Server Speed: 50       BEAUTIFUL DAY DRIVE SAFE
------------------------------
Waiting for Server response...
[CLIENT] Server sent:
[SERVER] instructions to Client[1]      You are close to next Vehicle 20.0  slow down         Signal strength okay    You are -10 faster than leader.Slow down!
 decelerate by -7.64  for the given time 1.31          Road Safe no object
---------------------------------
---------------------------------
```

Platoon faster than Main Vehicle

4. Object Detected: When Platoon updates the Main vehicle about object/external vehicle approaching the platoon, the Main vehicle instructs the Platoon to slow down and maintain the standard distance from the vehicle.

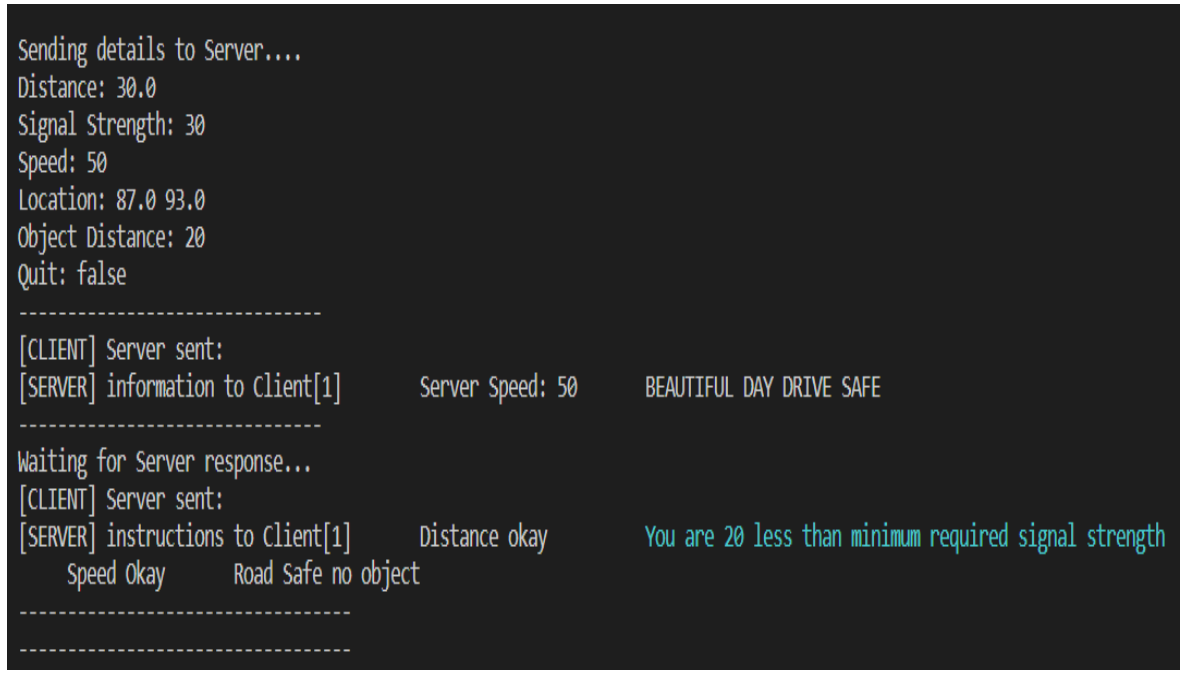
```
Sending details to Server....
Distance: 30.0
Signal Strength: 50
Speed: 50
Location: 57.0 -154.0
Object Distance: 10
Quit: false
------------------------------
[CLIENT] Server sent:
[SERVER] information to Client[1]       Server Speed: 50       BEAUTIFUL DAY DRIVE SAFE
------------------------------
Waiting for Server response...
[CLIENT] Server sent:
[SERVER] instructions to Client[1]      Distance okay          Signal strength okay    Speed Okay      You are 10.0  units
 from the Object, Slow down to maintain distance   20.0
---------------------------------
---------------------------------
```

Object detected

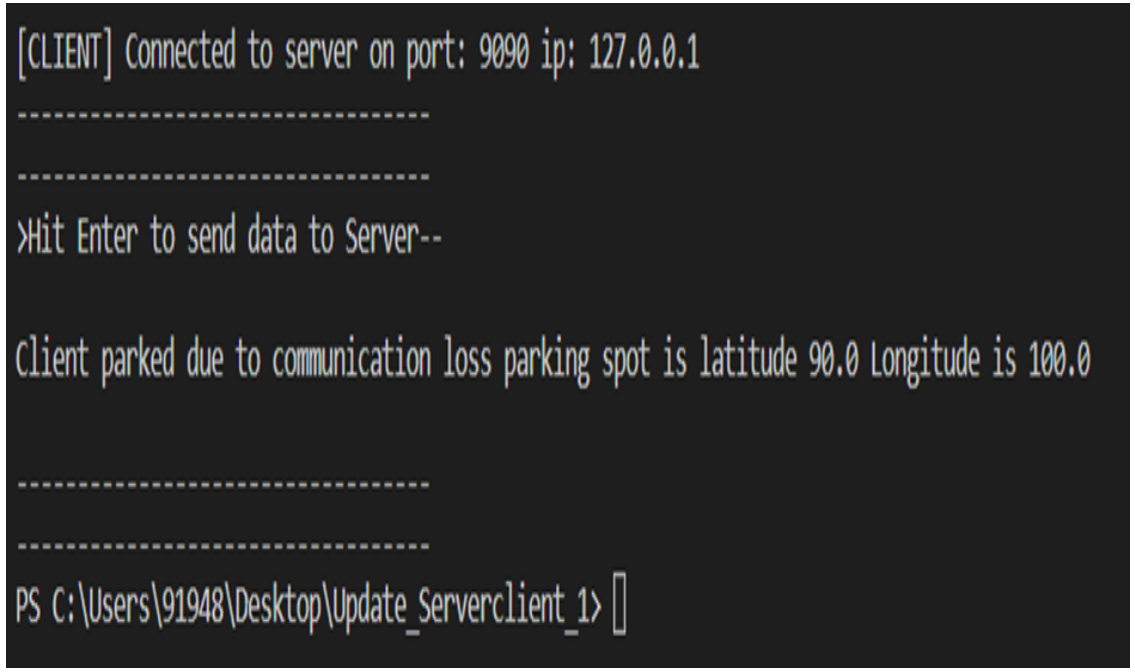
Results and Discussions

5. Signal Strength lost: When the signal strength is less than Main vehicle standards, it sends a warning message to the Platoon.

When the Platoon strength is less than standard signal strength expected by platoon to connect to the Main vehicle, it parks in the nearby available parking spot and updates the location.



Signal Strength less than Leader expectation Std.



Platoon parked itself and updated the location

Conclusion and Summary

The proposed system is successful in establishing the socket connections between one server socket and multiple client sockets using TCP protocol. The server socket a.k.a. Leader vehicle was able to monitor the following Platoons. The GPU programming was successfully implemented to monitor the speed of the platoons using OpenCL programming.

- What is Platooning
- Network Architecture and models
- Socket Programming
- GPU and JAVA implementation
- Working Scenarios


Thank You