

Truck Platooning System Design and Implementation

Roghieh Farajialamooti
Computer Science Department
Fachhochschule Dortmund
Dortmund, Germany
roghieh.farajialamooti001@stud.fh-dortmund.de

Behnoosh Hashemi Hendoukoshi
Computer Science Department
Fachhochschule Dortmund
Dortmund, Germany
behnoosh.hashemihendoukoshi001@stud.fh-dortmund.de

Ghazaleh Hadian Ghahfarokhi
Computer Science Department
Fachhochschule Dortmund
Dortmund, Germany
ghazaleh.hadianghahfarokhi001@stud.fh-dortmund.de

Danial Rafiee
Computer Science Department
Fachhochschule Dortmund
Dortmund, Germany
danial.rafiee001@stud.fh-dortmund.de

Ali Beiti Aydenlou
Computer Science Department
Fachhochschule Dortmund
Dortmund, Germany
ali.beitiaydenlou003@stud.fh-dortmund.de

Abstract—Truck platooning is expected to be an important the future intelligence transport system. Decreasing the workload of drivers, energy consumption, traffic congestions, as well as increasing road safety are the main benefits that have brought many research to this area.

This paper focuses on a four-truck platoon formation. We study movement behaviors, architecture, communication topology, requirements of the system in details. The implementation part of the system is developed by C++ programming language. Two different approaches of concurrent and parallel programming are applied in OpenMP and pthread on CPU to decline response time and computation time, because there are many tasks running in the ECU of each truck simultaneously. We implement GPU programming in CUDA in order to make use of GPU power for faster computations. We also present a client-server network by socket programming to simulate the data communication between trucks by networking four local devices.

Keywords—Truck platooning, Embedded systems, Parallelism, Concurrency, Real-time operations, socket programming, multithread programming

I. INTRODUCTION

The demand for freight transportation is increasing all over the world, the expansion of the road infrastructure is severely limited and the road sector shows high levels of greenhouse gases emission. Trucks transport about 70% of all domestic shipments. This enables businesses to move their products to other areas of the country and it also allows them to get the products and the supplies they need to continue running their companies[1]. 29% was Transportation's portion of 2016 petroleum-based energy consumption in the US. 10% of the US oil is consumed in long haul trucking annually. 38% of fleet operating expenses devoted to fuel. It has been reported that energy consumption costs constitute about 30% of the total operation cost of the truck industry. Trying to facilitate transportation and reduce the costs can be a great help and benefit in economy. Truck platooning is a solution here. A platoon is a formation of connected vehicles driving on the road with small inter-vehicular distances. Truck platooning is

the practice of virtually “coupling” two or more trucks in order to allow significantly shorter gaps between them. Vehicle platooning uses vehicle-to-vehicle communication to form and maintain a tight formation[9]. There are some autonomous functionalities such as electronically coupled braking or accelerating or even steering but each truck requires a driver remaining on the driving task and monitor the environment at all times. The main objective of the platoon formation is to reduce fuel consumption and carbon emissions, which achieve by “Drafting”. Drafting is the act of blocking airflow by creating low pressure in front of following vehicles. This leads to increase in fuel efficiency by reducing aerodynamic drag among a set of trucks. In fact, Aerodynamic drag is a force that the oncoming air applies to a moving body. It is the resistance offered by the air to the body's movement. So, when a truck is moving, it displaces the air. However, it affects the truck's speed and performance. Technically, it is the aerodynamic drag or the friction offered by the air to a vehicle. Therefore, a truck consumes less fuel in a platoon than moving alone. 6.7% of reduction in fuel consuming resulted from platooning. Congestion can also be reduced by fewer accidents and improved notification for drivers of congestion so they can choose an alternative route. As trucks can be driven automated and get benefits from sensing technologies like Vehicle-to-vehicle communication (V2V), Cameras, Radar (Radio detection and ranging), Lidar(light detection and ranging), localization via GPS (Global positioning system) and other sensor devices. V2V communication allows the following trucks to react to problems encountered by the lead truck much faster than a driver would be able to react. For example the following trucks do whatever the lead truck does. If it accelerates they will accelerate. If the lead brakes they will brake too. Due to the less usage of fuel, truck platooning has the potential to reduce CO2 emissions by up to 10% [3].

II. MOTIVATION

The rise in truck platooning-related research can be broadly divided into two categories: technique-related research and fuel-efficiency-related research. The fuel economy side principally focused on the precise savings realized by truck platooning and how to maximize the benefits, while the method side mostly focused on the engineering applications needed to make truck platooning a reality. The exact air drag and fuel consumption reduction from truck platooning, the coordination and optimization of the truck platooning formation, and the potential of truck platooning in a particular location are three key problems being studied in research based on fuel economy [4]. The reduction in fuel consumption were found to be 10% and 6%, respectively, for the leading and following trucks, with an inter-vehicle distance of 10 m, according to a two-truck test conducted by Browand et al. [2] on an empty airfield runway. The fuel consumption savings rate for the following truck and the leading truck both remain in the range of 10-12% when the inter-vehicle distance varies between 3 m and 10 m. Truck platooning holds great potential to make road transport safer, cleaner and more efficient in the future. Although there have been done many research on platoon formation in last decades, it is still a worthy problem to work on [6]. Some challenges still remain in this topic for instance the platoon formation of heterogenous vehicles or an adaptive scalable system being able to expand the number of vehicles is issues that have high potentials for further research. In this paper we design and implement a distributed controlling system. There is a network of four trucks that have a common destination. The followers chase a truck at the head of platoon and react and adapt to changes in its movement, requiring little to no action from drivers. While trucks are engaged on the network in auto driving mode, drivers will remain in control at all times, so they can also decide to leave the platoon and drive independently.

III. CONCEPTS

A. System requirements

In order to achieve the project objectives and practices and easier implementation, the following requirements have been specified and documented and validated first. The requirements are categorized in platoon behavior, communication behavior, design, failure strategy as shown in Figure 1.

- **Platoon behavior:** the platoon needs to make sure that every truck's behavior is synchronized.
 - Platoon formation: Trucks should be able to form a platoon within a specific range.
 - Distance maintenance: the platoon should be able to adjust to uniform/decoupling distance with neighboring trucks.
- Uniform distance: the platoon should be able to keep a uniform distance between each truck.
- Decoupling distance: the platoon should be able to create a gap.
- Position control: the platoon should be able to control trucks laterally and longitudinally.
- Driving: each truck should be able to drive manually or automatically.
- Obstacle avoidance: each truck should be able to detect surrounding objects and pass them.
- **Communication behavior requirements:** Each truck should receive data from the leading truck and built-in sensors consistently
 - Sensor data: each truck should receive the measuring values of sensors.
 - V2V data: each truck should receive a v2v data packet from the leading truck [8].
- **Design Requirements:** the system should be supplied with appropriate sensors and actuators to meet standard policy the platoon.
 - Sensing technologies: each truck should be equipped with multiple sensors such as camera, radar, lidar, and GPS and other required devices.
 - Controller: each truck should be equipped with an ECU including braking and steering systems for auto driving
 - V2V communication equipment: each truck should be equipped with a V2V module
- **Failure strategy requirements:** the platoon should be robust to potential failures.
 - Emergency stop: each truck should be able to stop in emergency situations to avoid accidents.
 - Communication failure: each truck should be able to reconnect the platoon system or use alternative data of sensors

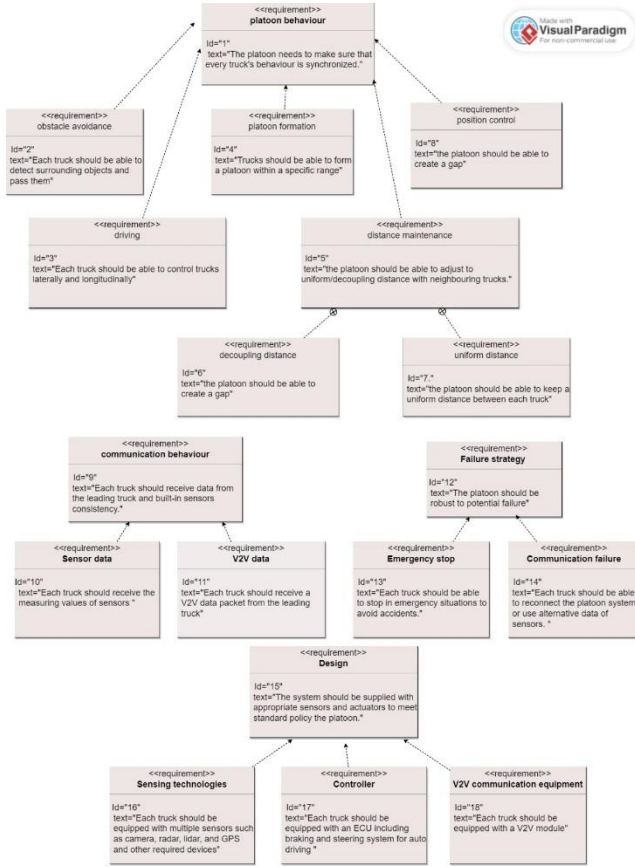


Figure 1: Requirement diagram

IV. SKETCH OF APPROACH

A. Use Case

When a driver assigns a leadership role to a truck, the platooning system is initialized. After that, trucks can ask to join the platooning system. As soon as the leader accepts the request, the followership is assigned to the truck and it is switched to auto-drive mode. If not in platooning mode, the driver in the followers truck can drive manually. The use case for the formation of the system is shown in Figure 2.

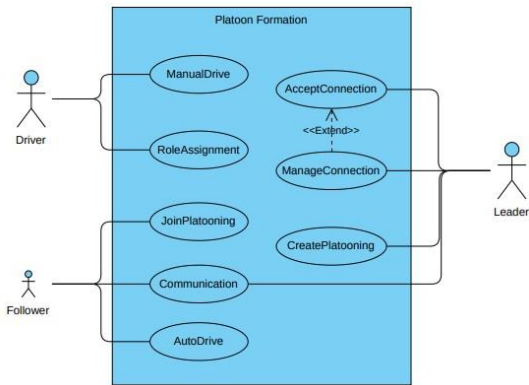


Figure 2: Platoon Formation

As shown, actors in the system are driver, leader and follower. Another use case for the system is to transform the role. As mentioned, a leader may quit the leadership and a

follower truck may become leader. As shown in Figure 3, when the driver of the leader truck asks to leave its role, the system transforms a follower to the leader. For that, the driver of the follower truck must accept the request.

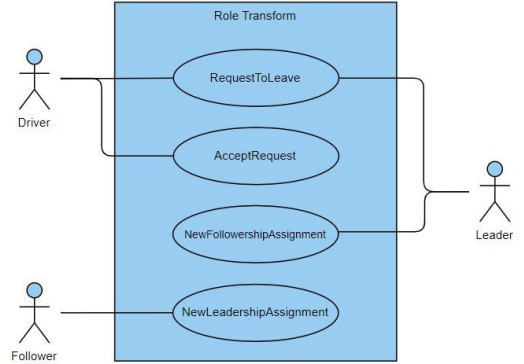


Figure 3: Use case for Role transformation

B. State Machine

The platoon can be stated in three main situations, which are leader, follower and idle. The state machine of a leader truck is shown in Figure 4.

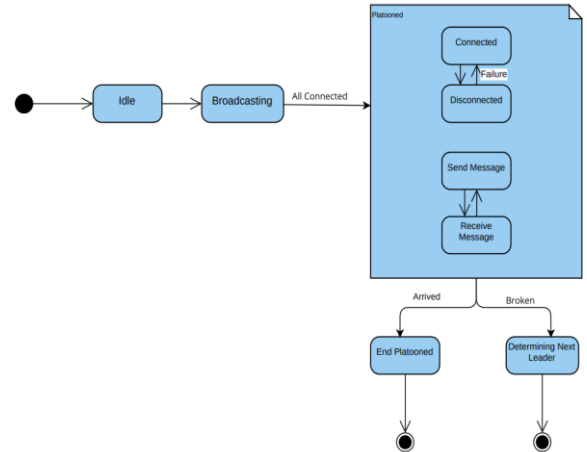


Figure 4: Leader state machine

The state machine of a following truck is drawn in Figure 5.

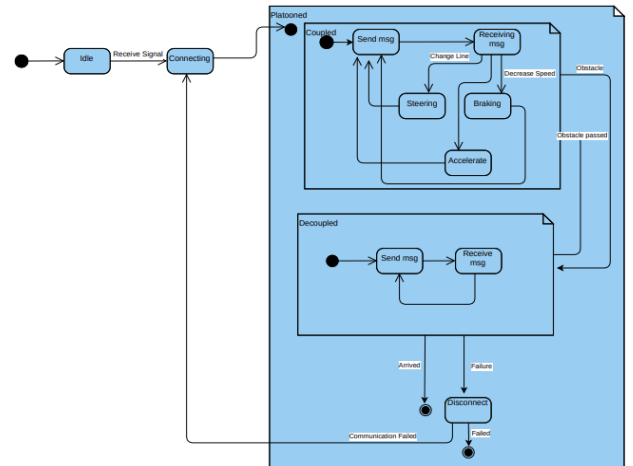


Figure 5: Follower state machine

C. Network Architecture

The platoon uses the semi-centralized peer-to-peer (p2p). It is a combination of Master-slave and peer-to-peer architecture, shown in Figure 6. The trucks are individual computing nodes and can process and compute by themselves [7]. At the beginning where there is no platoon shaped, one of the trucks sends a request message to become the leader or the super node (super peer). The other trucks receive this message and become the follower of the lead truck. All of the trucks get connected to the leader and receive data from the leader. Since the other nodes are independent computing units, they just receive the driving data like GPS coordinates, velocity, and the direction and process through the data and sends them to the auto driving system of the truck. This is the reason the nodes are peers not slaves. Because they have an independent controlling system and can act by the decisions made in their own controlling system. This system takes the advantages of both peer-to-peer and master-slave architecture. One of the advantages of this method is the failure in any node will not lead to a catastrophic consequence. For example, in master-slave architecture if the master confronts with a failure, the system will break and there isn't any automatic role change and the clients don't have the ability of becoming server. But in our architecture, when a failure occurs, the leader chooses the next truck as the next leader and transfers the role of leadership to the assigned truck. This is the most important advantage which is the capability of nodes to change role. It is flexible because the leader can choose the next leader. In the master slave architecture, the master assigns real time tasks to the slaves and failure in master will lead to failure in the whole system. Another of the advantage of this architecture is the independence of nodes. The nodes act individually for their system and are not dependent for deciding what to do. For example, the follower truck decides to accelerate or decelerate. To conclude, a semi-centralized peer-to-peer architecture is a flexible implementation for the platoon system, and is providing high level of node independence that makes the system robust against individual node failures.

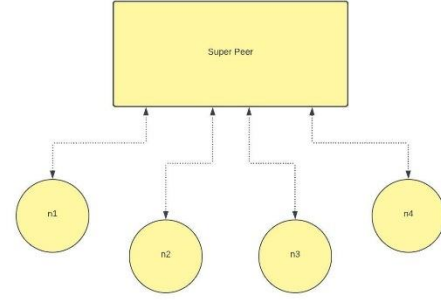


Figure 6: Network architecture of platoon

D. Layered Architecture

The truck platooning uses a layered architecture shown in Figure 7. Each layer handles the different functionalities of the system. First layer is the driving layer which has the functions of driving maneuver. Functions like, accelerating, braking, steering and so on. In other words, it handles the controlling of the vehicles. The Management layer is the layer in which the communication is handled. The communication between the leader and followers are in this layer and the data is transformed there. Also, the functions of decoupling, disengaging and engaging happen in the guidance layer. The Service layer focuses on the end-user side. This handles the new initiatives and the logistical operation demands. Hence, not much functionality is handled by this layer [5].

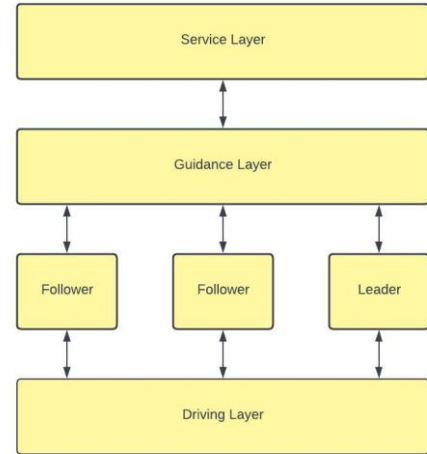


Figure 7: Layered architecture of platoon

E. UDP Communication

User Datagram Protocol (UDP) is a communication protocol that is primarily used to establish low-latency and loss-tolerating connections between applications on the internet. UDP speeds up transmissions by enabling the transfer of data before an agreement is provided by

the receiving party. As a result, UDP is beneficial in time-sensitive communications. Both UDP and TCP run on top of IP and are sometimes referred to as UDP/IP or TCP/IP. However, there are important differences between these two protocols. UDP sends messages, called *datagrams*, and is considered a best-effort mode of communications. This means UDP does not provide any guarantees that the data will be delivered or offer special features to retransmit lost or corrupted messages. User Datagram Protocol has attributes that make it beneficial for use with applications that can tolerate lost data. Because of the mentioned reasons UDP is a better protocol for platooning because it is much faster than TCP/IP. UDP allows packets to be dropped and received in a different order than they were transmitted, making it suitable for real-time applications where latency might be a concern. As a result, because of the latency of checking the messages by the TCP/IP and dropping the messages in case of receiving the message wrong, TCP/IP is not a suitable protocol for a hard-real-time system like platooning. to conclude, UDP is the protocol that is used because it is fast, best effort, more efficient and suitable for a real time system.

F. Communication Failure

Communication is about data sharing. Any loss of data through either V2V communication or sensors in each truck can be considered as communication failure. In case of failure, the appropriate actions need to be taken to ensure the robustness of our system. Generally, two types of failures are considered in this project:

- 1- V2V Communication Failure: if this happens in the following truck, it leads to loss of critical data including GPS position, acceleration and so on from the leading truck. Since it is a major failure, it will highly impact platooning. In fact, the truck will disconnect from the platooning network and in turn the autonomous driving mode will exit. With respect to failure in the lead truck, it also will disengage from the system and we aim to assign the role of leader to another following truck to lead the others. In case of unsuccessful substitution of leader role, the whole platoon will stop, so the vehicles can brake and stop on the road.
- 2- Loss of sensing data: it affects the quality of auto-driving in each truck. To address this issue, the ECU makes use of alternative data from other sensors.
 - Loss of radar data: Loss of Distance & relative velocity data between the trucks. It is a minor

failure. This data can be retrieved from cameras and V2V communication.

- Loss of camera data: this means the system cannot distinguish the type of surrounding objects. Although this data helps for better auto driving, we have alternative sensors, radar and Lidar. So, it is considered as a minor failure in the system.
- Loss of lidar data: it means the truck lost a high-quality 3D image of objects around which is used in automated driving. Although access to this data plays an important role in driving, the ECU needs to handle this deficiency by using alternative data from Camera and Radar.

In Figure 8, the activity diagram in case communication failure situations is shown.

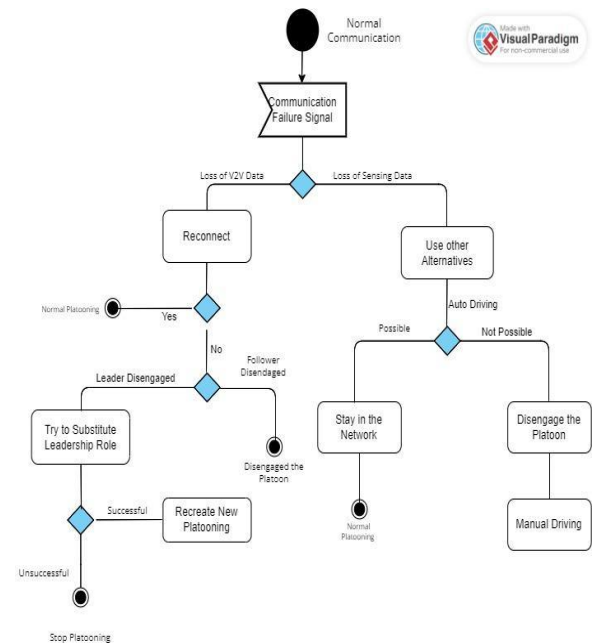


Figure 8: Communication failure activity diagram

V. IMPLEMENTATION

The implementation of the system is developed in C++. We apply parallel concurrent programming using two API platforms OpenMP and pthread. Let's talk about parallelism and concurrency in the following section.

A. Parallelism and Concurrency

In this project, there are two different approaches to implement the computational parts of the system; concurrency and parallelism. Concurrency is about multiple independent tasks which start, run, and complete in overlapping time periods, in no specific order. It is an approach that is used for decreasing the response time of the system by using the single processing unit. With regards to the truck platooning system, since we deal with a real time system in which multiple independent functions occur in each truck, it would be more

efficient to take advantage of the aforementioned technologies. To be more specific, the following trucks are driven autonomously, a lot of concurrent tasks are running on each follower node to control its various driving parameters based on the current state of the leader truck. Each concurrent task is executed on a separate thread of execution. For example, there is the steering thread which is responsible for regularly adjusting the steering of the follower node according to the leader truck or the speed Control thread which is responsible for adjusting the speed of the follower node according to the current leader speed and try to maintain the speed within a certain limit. Similarly with other threads like braking thread etc. In the execution phase, there are two known application interfaces (API) named pthread and OpenMP which represent two totally different multiprocessing paradigms. pthread and OpenMP used for concurrent and parallel programming, respectively. We used OpenMP to implement parallelism in our platoon scenario. Tasks that can be decomposed into independent subtasks are executed in parallel on separate threads to finish them faster. An OpenMP program has sections that are sequential and sections that are parallel. In general, an OpenMP program starts with a sequential section in which it sets up the environment, initializes the variables, and so on. When run, an OpenMP program will use one thread (in the sequential sections), and several threads (in the parallel sections). There is one thread that runs from the beginning to the end, and it's called the master thread. The parallel sections of the program will cause additional threads to fork. These are called the slave threads.

B. GPU Programming

General-purpose computing on a GPU, better known as GPU programming, is the use of a GPU together with a CPU to accelerate computation in applications traditionally handled only by the CPU. Even though GPU programming has been practically viable only for the past two decades, its applications now include virtually every industry. For example, GPU programming has been used to accelerate video, digital image, and audio signal processing, scientific computing, computer vision, neural networks and deep learning, among many other areas.

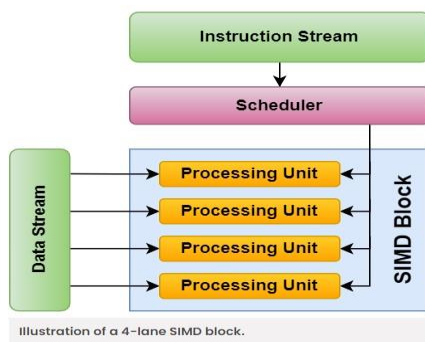


Figure 9: GPU architecture

A CPU consists of a few cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously. In this project, as each truck is equipped with multiple sensors, including camera, radar, Lidar, GPS and so on, and measuring the values of these devices in each truck accrues real time and simultaneously, we take advantage of GPU technology for a higher level task computation. The kernel assigned to the GPU will describe the task that should be performed by each thread, which basically consists of two steps: reading the sensor value and storing it in the array of readings. There are many parallel computing platforms such as CUDA, OpenCL, OpenACC to harness the power of GPU. In this project, CUDA alongside C++ programming language is used.

C. Result of Code Execution

The system supports both concurrent and parallel programming. The main class is the truck which has four public attributes: id - which is from one to zero -, role - which can be either leader or follower -, velocity and direction. The direction is defined by three integers 0, 1, and 2. 0 is when the leader truck is driving straight forward, 1 is when it turns to the right and 2 is when it turns to the left.

```

private:
    int id; // 0 to 3 for four trucks

public:
    bool role; // 1 if leader, 0 if follower
    int velocity;
    int direction; // 1 if ahead, 2 if right, 3 if left
    
```

Figure 10: Class truck attributes

Leader and follower trucks use the same methods for assigning roles and getting input from external devices such as radar, lidar, GPS, and sensors. Sending messages and accepting connections are specified for the leader truck. In addition, sending connection requests, receiving data, and driving functions such as steering, accelerating and braking are specified for follower trucks. The truck brakes if the velocity received by the leader is greater than its own velocity, and accelerates otherwise.

```

Truck(int iD) { id = iD; } // constructor
void assignLeadership() { role = 0; };
void assignFollowership() { role = 1; };
static void sensor();
static void radar();
static void lidar();
static void gps();

// leader methods
static void acceptConnectionReq();
void sendData();

// Follower methods
static void sendConnectionReq();
void receiveData();
void steering(Truck);
void accelerate(Truck);
void brake(Truck);

```

Figure 11: Class truck methods

As for the tasks, we assign three threads for driving, communication and inputs received from external devices. These tasks are done all the time. The driving function itself consists of three parallel tasks. One for steering, one for accelerating and one for system brake. As a result of the truck's role, thread communication parallels its tasks. So, two parallel tasks are running for sending and receiving data. The third thread is dedicated to external input gathered from devices.

```

thread t1(thread_drive);
thread t2(thread_communication);
thread t3(thread_input);

t1.join();
t2.join();
t3.join();

return 0;

```

Figure 12: Threads

Parallelism is done by OpenMP. To do so, we need to include the OpenMP header to the program. The following code is an example of thread communication, and contains two parallel tasks of sending and receiving data.

```

void thread_communication()
{
    while (true)
    {
        #pragma omp parallel num_threads(2)
        {
            #pragma omp task
            {
                truck0.sendData();
            }

            #pragma omp task
            {
                truck0.receiveData();
            }
        }
    }
}

```

Figure 13: Parallelization on communication

So, the output of the system would be as follows. The tasks are running in a while loop, and they sleep for 1000 milliseconds after each run.

```

Truck 1 is Accelerating
drive ahead
drive ahead
Connected
drive ahead
Truck 2 is Accelerating
drive ahead
drive ahead

```

Figure 14: The output of the system

In order to communicate between trucks, UDP is used. The message sent by the leader contains the velocity and direction. By comparing the velocity of the leader to that of its own, the follower parses the message and decides if it accelerates or breaks. Steering is controlled by the direction sent in the message.

```

SOCKET out = socket(AF_INET, SOCK_DGRAM, 0);

// wait out to the socket

string v(argv[1]); // get the velocity
int sendOk = sendto(out, v.c_str(), v.size() + 1, 0, (sockaddr *)&server, sizeof(server));

```

Figure 15: Sending datagram to the client's socket

Another way of parallelization is through the GPU. CUDA, designed by NVIDIA, was used in our case. To do so, Colab environment was chosen to implement CUDA. It must be noted that the runtime needs to be set on the GPU. In order to install CUDA, the command in Figure 16 was used.

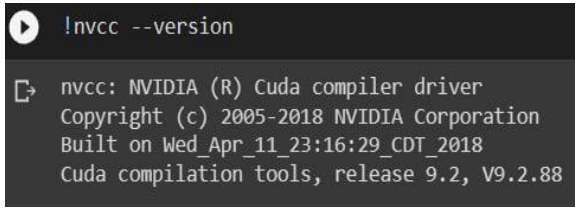
```

!apt-get --purge remove cuda nvidia* libnvidia-*
!dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 dpkg --purge
!apt-get remove cuda-*
!apt autoremove
!apt-get update

```

Figure 16: Installing CUDA

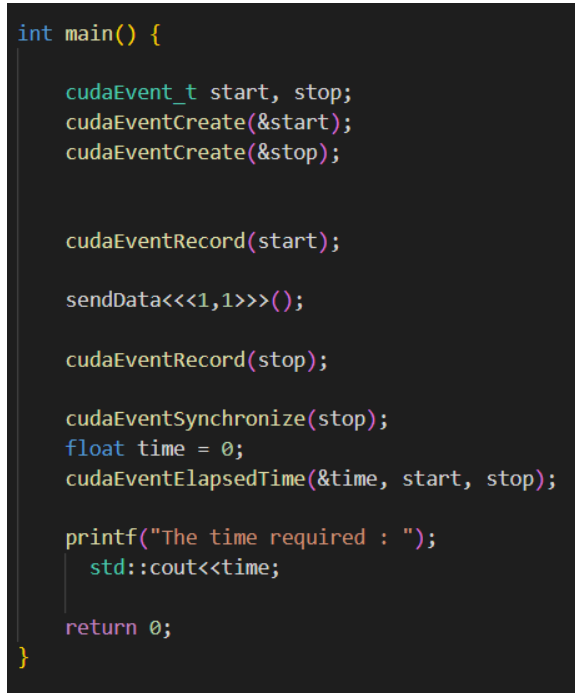
It must be noted that the runtime needs to be set on the GPU. To be able to use CUDA, %%c must be added to the first line of the code.



```
!nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2018 NVIDIA Corporation
Built on Wed Apr 11 23:16:29 CDT 2018
Cuda compilation tools, release 9.2, V9.2.88
```

Figure 17: Coda version

The method sending data by the leader is run on the GPU. The timestamp according to run the method is shown in Figure 18.



```
int main() {
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    cudaEventRecord(start);

    sendData<<<1,1>>>();

    cudaEventRecord(stop);

    cudaEventSynchronize(stop);
    float time = 0;
    cudaEventElapsedTime(&time, start, stop);

    printf("The time required : ");
    std::cout<<time;

    return 0;
}
```

Figure 18: A method run in GPU

The time required to do so is 0.1269 milliseconds; compared to the runtime on CPU which was 0.001.

VI. SUMMARY

Platoon formation of truck has drawn the research attentions and projects, since it contributes to more efficient transportation in the world. The main challenge is the designed system should be robust, reliable, safe and fast in

the real-time situation. The platoon should adapt with the real situations of the roads next to other vehicles. This project consists of the design and implementation of a set of four trucks which aim to form a platoon. The requirements of the system in different aspects are analyzed. For data transmission, we consider V2V communication between nodes. The system is developed in C++ in two modes of CPU and GPU programming. For CPU implementation, we apply multithread programming to decrease the response time of running the tasks and have a better control of vehicle in auto driving mode.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to Prof. Dr. Stefan Henkler, for their invaluable guidance and support in the completion of this project.

REFERENCES

- [1] Liang, K.-Y.; Mrtensson, J.; Johansson, K.H. "Fuel-saving potentials of platooning evaluated through sparse heavy-duty vehicle position data." In Proceedings of the 2014 IEEE Intelligent Vehicles Symposium Proceedings, Dearborn, MI, USA, 8–11 June 2014; pp. 1061–1068.
- [2] Browand, F.; McArthur, J.; Radovich, C. "Fuel Saving Achieved in the Field Test of Two Tandem Trucks; Institute of Transportation Studies" at UC Berkeley: Berkeley, CA, USA, 2004.
- [3] S. Ellwanger and E. Wohlfarth, "Truck platooning application," 2017 IEEE Intelligent Vehicles Symposium (IV), 2017.
- [4] Han, Y.; Kawasaki, T.; Hanaoka, S. The Benefits of Truck Platooning with an Increasing Market Penetration: A Case Study in Japan. Sustainability 2022, 14, 9351.
- [5] R. Gheorghiu, V. Iordache, and A. C. Cormos, "Cooperative Communication Network for Adaptive Truck Platooning," researchgate, 2017.
- [6] Janssen, M. "Truck Platooning: Driving The Future of Transportation", TNO white paper, 2015
- [7] James F. Kurose , Keith W. Ross "Computer Networking: A Top-Down Approach"– 5 Mar. 2012
- [8] B. Masini, A. Bazzi, and A. Zanella, "A survey on the roadmap to mandate on board connectivity and enable V2V-based vehicular sensor networks," Sensors, vol. 18, no. 7, p. 2207, 2018.
- [9] A. Rao, A. Sangwan, A. A. Kherani, A. Varghese, B. Bellur, and R. Shorey, "Secure V2V Communication With Certificate Revocations," in Mobile Networking for Vehicular Environments, May 2007, pp. 127–132.

Code Implementation of four files: main.cpp, server.cpp, client.cpp, gpu.cpp

/* Truck Platooning System */

/* main.cpp */

```
#include <iostream>
#include <string>
#include <thread>
#include <omp.h>

using namespace std;

class Truck
{
private:
    int id; // 0 to 3 for four trucks

public:
    bool role; // 1 if leader, 0 if follower
    int velocity;
    int direction; // 1 if ahead, 2 if right, 3 if left

    Truck(int iD) { id = iD; } // constructor
    void assignLeadership() { role = 0; };
    void assignFollowership() { role = 1; };
    static void sensor();
    static void radar();
    static void lidar();
    static void gps();

    // leader methods
    static void acceptConnectionReq();
    void sendData();

    // Follower methods
    static void sendConnectionReq();
    void receiveData();
    void steering(Truck);
    void accelerate(Truck);
    void brake(Truck);
};

void Truck::steering(Truck t)
{
    if (t.direction == 0)
    {
        cout << "Drive ahead" << endl;
    }

    if (t.direction == 1)
    {
        cout << "Drive right" << endl;
    }

    if (t.direction == 2)
    {
        cout << "Drive left" << endl;
    }
}

void Truck::accelerate(Truck t)
{
    if (t.velocity < received.velocity)
    {
        cout << "Truck " << id << "Is Accelerating";
    }
}

void Truck::brake(Truck t)
{
    if (t.velocity > received.velocity)
    {
        cout << "Truck " << id << "Is Breaking";
    }
}

void Truck::sendData()
{
    if (this->role == 0)
    {
        cout << "Sent data is " << endl;
    }
}

void Truck::receiveData()
{
    if (this->role == 1)
    {
        cout << "Received data is " << endl;
    }
}

Truck truck0(0); // name differently on each system

void thread_drive()
```

```

{

    while (true)
    {

#pragma omp parallel num_threads(3)
    {

#pragma omp task
    {

        truck0.accelerate(truck0);
        this_thread::sleep_for(1000ms);

    }

#pragma omp task
    {

        truck0.brake(truck0);
        this_thread::sleep_for(1000ms);

    }

#pragma omp task
    {
        truck0.steering(truck0);
        this_thread::sleep_for(1000ms);
    }
    }

}

void thread_communication()
{

    while (true)
    {

#pragma omp parallel num_threads(2)
    {

#pragma omp task
    {

        truck0.sendData();
        this_thread::sleep_for(1000ms);

    }

#pragma omp task
    {

        truck0.receiveData();
        this_thread::sleep_for(1000ms);

    }
    }

}

void thread_input()
{

    while (true)
    {

#pragma omp parallel num_threads(4)
    {

#pragma omp task
    {
        Truck::sensor();
        this_thread::sleep_for(1000ms);
    }

#pragma omp task
    {
        Truck::radar();
        this_thread::sleep_for(1000ms);
    }

#pragma omp task
    {
        Truck::lidar();
        this_thread::sleep_for(1000ms);
    }

#pragma omp task
    {
        Truck::gps();
        this_thread::sleep_for(1000ms);
    }
    }

}

int main()
{

    truck0.assignLeadership();

    if (truck0.role == 1)
    {
        Truck::acceptConnectionReq();
    }
}

```

```

    }

    if (truck0.role == 0)
    {
        Truck::sendConnectionReq();
    }

    thread t1(thread_drive);
    thread t2(thread_communication);
    thread t3(thread_input);

    t1.join();
    t2.join();
    t3.join();

    return 0;
}

```

/* Impelemnting UDP Server System */

/* server.cpp */

```

#include <iostream>
#include <ws2tcpip.h>
#include <omp.h>

using namespace std;

#pragma comment(lib, "Ws2_32.lib")

int main()
{
    // Initialize Winsock

    WSADATA wsaData;
    WORD version = MAKEWORD(2, 2);

    int wsok = WSASStartup(version, &wsaData);
    if (wsok != 0)
    {
        cout << "Can't Start Winsock " << wsok << endl;
        return 0;
    }

    // Bind socket to ip address and port

    SOCKET in = socket(AF_INET, SOCK_DGRAM, 0);
    sockaddr_in serverHint;
    serverHint.sin_addr.S_un.S_addr = ADDR_ANY;
    serverHint.sin_family = AF_INET;
    serverHint.sin_port = htons(54000); // port

    if (bind(in, (sockaddr *)&serverHint, sizeof(serverHint)) == SOCKET_ERROR)
    {
        cout << "Can't bind socket " << WSAGetLastError() << endl;
        return 0;
    }

    sockaddr_in client;
    int clientLenght = sizeof(client);
    ZeroMemory(&client, clientLenght);

    char buf[1024];

    // Enter a loop
    while (true)
    {
        ZeroMemory(buf, 1024);

        // Wait for message

        int bytesIn = recvfrom(in, buf, 1024, 0, (sockaddr *)&client, &clientLenght);
        if (bytesIn == SOCKET_ERROR)
        {
            cout << "Error from client " << WSAGetLastError() << endl;
            continue;
        }

        // Display message

        char clientIp[256];
        ZeroMemory(clientIp, 256);

        inet_ntop(AF_INET, &client.sin_addr, clientIp, 256);

        cout << "Message received from " << clientIp << " : " << buf << endl;

        if (bytesIn == 0)
        {
            cout << "Brake!" << endl;
        }
    }

    // Close socket

    closesocket(in);

    // Shutdown Winsock

```

```

    WSACleanup();
}

```

/* Implementing UDP Client System */

```
/* client.cpp */
```

```

#include <iostream>
#include <ws2tcpip.h>
#include <string>
#include <omp.h>

```

```
using namespace std;
```

```
#pragma comment(lib, "Ws2_32.lib")
```

```

int main(int argc, char *argv[])
{

```

```
    // Initialize Winsock
```

```

    WSADATA wsaData;
    WORD version = MAKEWORD(2, 2);

```

```

    int wsok = WSASocket(VERSION, SOCK_DGRAM, 0, NULL, 0);
    if (wsok != 0)
    {
        cout << "Can't Start Winsock " << wsok << endl;
    }

```

```
    // Create a hint structure for server
```

```

    sockaddr_in server;
    server.sin_family = AF_INET;
    server.sin_port = htons(54000);

```

```
    inet_pton(AF_INET, "127.0.0.1", &server.sin_addr);
```

```
    // Create socket
```

```
    SOCKET out = socket(AF_INET, SOCK_DGRAM, 0);
```

```
    // Wait out to the socket
```

```

    string v(argv[1]); // get the velocity
    int sendOk = sendto(out, v.c_str(), v.size() + 1, 0, (sockaddr *)&server, sizeof(server));

```

```

    if (sendOk == SOCKET_ERROR)
    {
        cout << "Didn't work " << WSAGetLastError() << endl;
    }

```

```
    // Close socket
```

```
    closesocket(out);
```

```
    // Shutdown Winsock
```

```

    WSACleanup();
}

```

/* Implementing a Method on GPU */

```
/* gpu.cpp */
```

```

%%cu
#include <cstdio>
#include <iostream>

```

```

__global__ void sendData(){
    printf("Send Velocity and Direction \n");
}

```

```
int main() {
```

```

    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

```

```
    cudaEventRecord(start);
```

```
    sendData<<<1,1>>>();
```

```
    cudaEventRecord(stop);
```

```

    cudaEventSynchronize(stop);
    float time = 0;
    cudaEventElapsedTime(&time, start, stop);

```

```

    printf("The time required : ");
    std::cout<<time;

```

```

    return 0;
}

```

AFFIDATIV

We “Roghieh Farajialamooti”, “Behnoosh Hashemi Hendoukoshi”, “Ghazaleh Hadian Ghahfarokhi”, “Danial Rafiee”, “Ali Beiti Aydenlou” herewith declare that we have composed the present paper and work by ourselves and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form has not been submitted to any examination body and has not been published. This paper was not yet, even in part, used in another examination or as a course performance.

Dortmund, 19.02.2023

Roghieh Farajialamooti



Behnoosh Hashemi Hendoukoshi



Ghazaleh Hadian Ghahfarokhi



Danial Rafiee



Ali Beiti Aydenlou



GitHub project link: <https://github.com/Truck-PJT/Truck/tree/main/final>