Truck Platoon System

Adijat Ajoke Sulaimon
Adijat.sulaimon001@stud.fh-dortmund.de
Ammar Haziq
ammar.binmohdhalim003@stud.fh-dortmund.de
Zahra Mahdion
Zahra.mahdion001@stud.fh-dortmund.de
Amir Hossein Pakdel
Amir.pakdel001@stud.fh-dortmund.de

***Abstract*—In this project, a truck platoon system is developed and implemented based on a distributed and parallel system concept, central to Embedded Systems Engineering (ESE). The system employs a server-client architecture, utilizing TCP/IP for reliable and efficient communication between platoon members. It consists of a leader truck, follower trucks, and external vehicles (such as cars) interacting with the platoon. A key scenario addressed is the dynamic adaptation of the platoon when a car attempts to pass between the trucks. The system enables real-time platoon reconfiguration, ensuring safe vehicle interactions while maintaining stability and efficiency. This is achieved through a structured communication protocol that allows continuous data exchange regarding speed, distance, and formation adjustments. Additionally, the project explores distributed architectures and parallel programming techniques, core elements of ESE, to optimize performance. The system, developed in C++, leverages its computational efficiency for real-time processing. By incorporating parallel programming strategies and real-time data management, the platoon system ensures low-latency communication and quick decision-making, crucial for autonomous vehicle coordination. Implementation results, observed through console outputs and graphical simulations, demonstrate the system's effectiveness in handling dynamic traffic interactions, offering a promising solution for modern logistics and autonomous fleet management in the context of embedded systems.**

Keywords— Leader Truck, Follower Trucks, Passing Vehicle, TCP/IP, Server-Client Architecture, C++, Platoon Reconfiguration

I. Introduction

Truck platooning is an advanced technology that allows multiple trucks to move together in a coordinated manner, improving safety, efficiency, and fuel savings. In our system, a leader truck, follower trucks, and other vehicles such as passing cars interact using a server-client architecture with TCP/IP communication [1]. Unlike autonomous platooning, where all trucks drive independently, our system focuses on real-time communication and coordination between the trucks to maintain a stable platoon formation [2]. One key scenario we address is when a car attempts to pass through the platoon. This requires the system to dynamically adjust the truck spacing while maintaining safe and smooth operations. The system continuously monitors speed, distance, and formation changes, allowing trucks to react immediately to surrounding traffic conditions.

Instead of traditional Vehicle-to-Vehicle (V2V) communication, we use TCP/IP, which is a reliable and widely used network protocol. This enables trucks to exchange real-time data such as speed, direction, and emergency signals using socket-based communication. TCP/IP also ensures secure and stable connections, reducing risks of cyber threats and data loss.

Our system is implemented in C++, allowing for high-performance computing and real-time responsiveness. The results are observed through console outputs and graphical simulation in TinkerCad, demonstrating how the platoon reacts in different scenarios. By improving truck coordination and traffic interaction, our project contributes to smarter and more efficient fleet management.

II. Motivation

The logistics and freight industry faces many challenges, such as high fuel costs, road safety issues, and managing large fleets efficiently. As companies look for ways to make transportation cheaper, safer, and more reliable, truck platooning has emerged as a promising solution [2] [1]. Truck platooning allows multiple trucks to drive closely together in a coordinated way, reducing fuel consumption and carbon emissions by cutting down air resistance. It also improves road safety by ensuring that trucks move together smoothly, reducing the chances of accidents caused by sudden braking or lane changes [1]. However, real-world platooning comes with its own set of challenges. Keeping a safe and flexible distance between trucks while adjusting to road and traffic conditions. Ensuring smooth communication between trucks so they can react quickly to any changes. Handling situations where a car tries to pass through the platoon, which requires the trucks to make real-time adjustments. Creating a system that works efficiently without relying on one central controller, so if one truck has an issue, the others can still function properly. Our project focuses on building a server-client-based truck platooning system using TCP/IP communication and C++ programming. This system allows trucks to adjust their positions dynamically, making the platoon more flexible and adaptable to real-world driving conditions. When a car tries to pass through the platoon, our system ensures the trucks react in a safe and controlled manner.

By using distributed parallel and embedded systems, our project provides a smart and scalable solution for logistics companies. This can help businesses save fuel, improve road safety, and automate their truck fleets. The project also supports the development of future smart transportation systems, helping trucks communicate better with each other and with surrounding traffic. Overall, this project makes trucking more efficient, safer, and better for the environment, while also

preparing for the future of autonomous vehicles and intelligent transport systems.

III. Design & System Architecture

The Design and System Architecture section provides a comprehensive overview of the truck platooning system's structural and behavioral models. It includes all relevant SysML and UML diagrams, offering both a visual and technical representation of the system's design. These models illustrate the system components, interactions, constraints, and operational behavior, ensuring safe and efficient platooning.

A. Requirement Diagram

The Requirement Diagram provides a structured overview of the system's functional and non-functional requirements. It serves as a foundation for our system design, development, and validation, ensuring that every group member have a common understanding of our system's expectations.The diagram in figure 1, outlines the key functionalities of the truck platooning system.

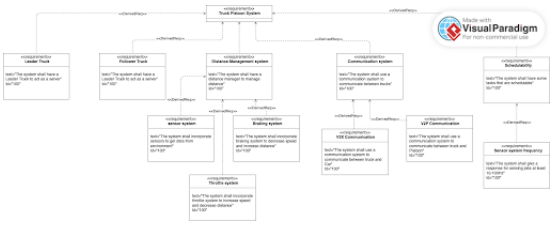

Fig. 1. Requirement Diagram

B. Use Case Diagram

In our project we have 3 Use Cases, here we will discuss about all three of them. In the first use case in figure 2, we have three actors. Leader (Truck with driver), Follower (Driverless Truck) and Car (Other Road Users). Leader continuously monitors platoon status. Car asks if it can pass through platoon, then the Follower Detect it and increase the distance to make safe distance in platoon for Car to pass and then decreases the distance to make the platoon in correct distances.

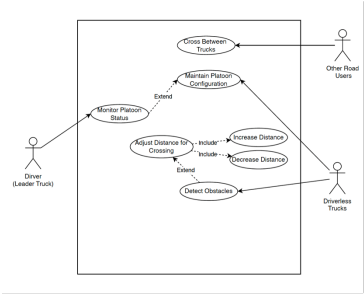

Fig. 2. Use Case Diagram

In the second Use Case in figure 3, we have another scenario. In this Use Case we have two actors. Leader (Truck with driver) and Follower (Driverless Truck). When Follower truck wants to join the platoon, it sends a message to Leader and ask if it can join to the platoon. The Leader can Accept or Reject the request. If it accepts, the Follower could join the platoon and if rejects, the Follower could not join the platoon.

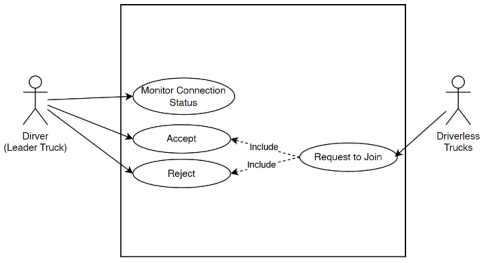

Fig. 3. Use Case Diagram

In the last Use Case, we have two actors. Leader (Truck with driver), Follower (Driverless Truck). Here the most important thing is Synchronize Movement which leads by the Leader. Here all Follower should synchronize their Speed and Direction with the Leader truck.

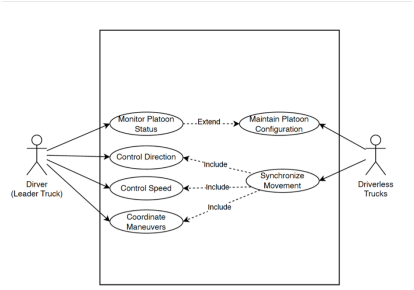

Fig. 4. Use Case Diagram

C. Sequence Diagram

After we identified and designed the use cases, now we move on to the design of the sequence diagram. Sequence diagrams functions as to model the interaction of message passing between the different entities that we consider fort the truck platooning system. The scenario that we have chosen for the sequence diagram is based on the scenario that a car entity wants to pass the platoon. One can imagine that this scenario occurs when a car wants to make an exit from the highway, but the platoon is in the way. Therefore, the platoon needs to make space for the car to pass.

Based on the figure 5. We have identified three main actors for the sequence diagram. A car, a leader truck and a follower truck. Firstly, the car will make a request to the leader, indicating that it wants to pass the platoon. The leader will then send a car passing command to the follower truck in order for it to increase the distance. Then, the follower will internally execute the command and increase its distance from the leader

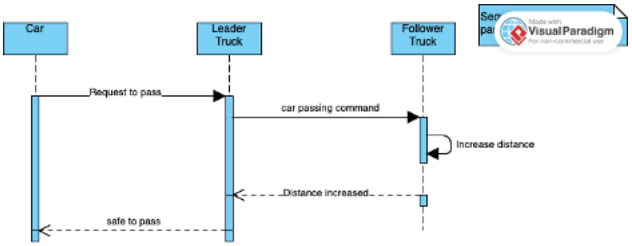

Fig. 5. Sequence Diagram

truck. Once the command has been executed, the follower will reply to the message saying that it has increased its distance. Lastly, the leader truck will reply to the initial request from the car indicating that it is now safe to pass the platoon. Our plan is for the future, we will expand this sequence into the car passing the platoon, send a message to the leader that it has passed and the leader will give a command to the follower to decrease its distance.

D. Block Diagram

In realizing the architecture for the system, we first developed a block diagram to identify main components in blocks for the system that we want to have. The figure 6 describes the block diagram that we have finalized. The truck platooning system shall contain the main block of trucks. Generalization of instances of trucks can be either Leader Truck or Follower truck. Each truck block will contain the blocks User interface, Communication system, and distance management system. Examples of the user interface could be steering wheel, throttle, or brake to name a few. The communication system block will have relations with one V2X communication block and one V2P communication block. The distance management system will have relations to a control unit that will perform computations and decision making, Actuator system that consists of throttle system and braking system, to control the acceleration and deceleration thus having control of the distance between trucks, and lastly a sensor block that is in charge of gathering data from the environment. The sensor block may consist of radars for an example. With the addition of this block diagram it helps the developers to understand the structure of the system by identifying the main components of the system. Next we will look at the parametric constraints diagram to visualize the constraints of the system and internal block diagrams to give more details on how the blocks interact with each other.

E. Node specific Architecture

We have developed a node specific architecture for the different entities of our system. The reason for this is we wanted to flesh out the abstraction of the system architecture of each entity without going into the direction of internal block diagrams. The node specific architecture can be observed in the figure 7. Each node will contain a communication layer to handle the communication between different nodes. The communication layer will handle the communication between

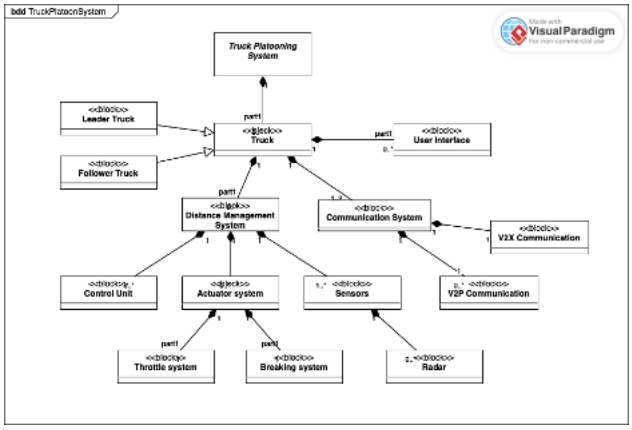

Fig. 6. Block Diagram

each node via a communication protocol. The message queue will capture and store messages that will either be received or transmitted. The behavior controller handles the messages and perform the needed actions based on the messages from the message queue and put the back to the message queue to be send. It is worth noting that the arrows represent the data flow which in this scenario of our system represents the messages that are being passed by the different entities. Another detail that we will discuss in the implementation section is that there will be different queues for receiving and transmitting but it is not needed in the diagram as to serve an abstraction of the said detail.

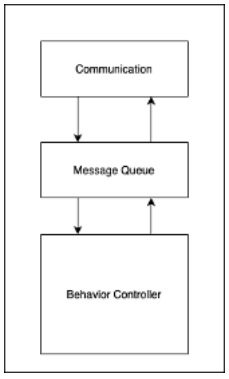

Fig. 7. Node specific Architecture

F. State Machine Diagram

The state machine diagram in figure number 8 represents the behavior of our truck platoon system and how it dynamically adjusts when a car attempts to pass through the platoon. This model ensures that the platoon maintains safe distances, smooth transitions, and efficient communication between trucks.

1. Initial State: DrivingState Our system starts in the DrivingState, where the platoon operates under normal conditions. The trucks maintain a safe following distance and communicate with each other to stay synchronized. This is the

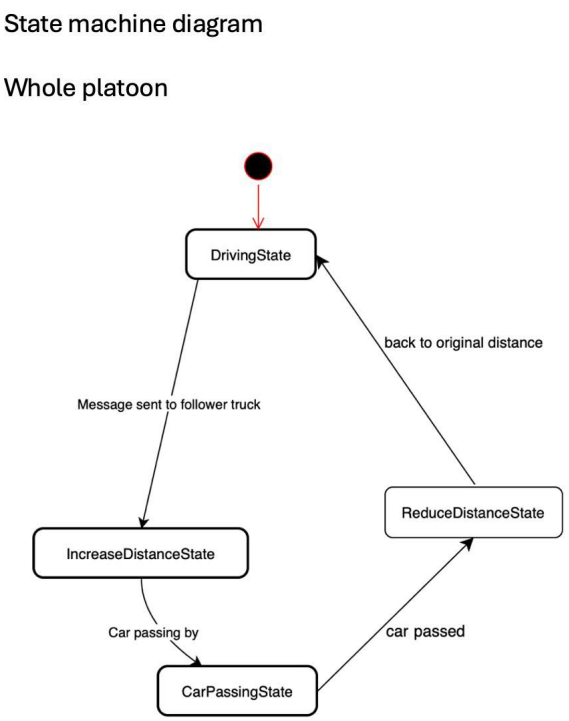

Fig. 8. State Machine Diagram

default state of the system, ensuring that the trucks are driving efficiently without any external interruptions. 2. Transition to IncreaseDistanceState When the system detects that a car wants to pass between the trucks(Car send a message that i want to pass), the leader truck sends a message to the follower trucks to increase their distance. This step is necessary to create enough space for the passing vehicle, preventing sudden braking or unsafe lane changes. 3. IncreaseDistanceState (Making Space for the Car) At this stage, the trucks gradually increase their distance from each other, allowing enough room for the car to merge safely. This ensures a smooth and controlled reaction to external traffic without disrupting platoon stability. The system remains in this state until the car is actively passing through the platoon. 4. Transition to CarPassingState Once the car starts moving between the trucks, the system transitions to the CarPassingState. In this state, the trucks hold the increased distance, allowing the car to pass safely without interfering with the platoon's structure. 5. Transition to ReduceDistanceState After the car has passed through and exited the platoon (It can be detected by sensors also ncar send message that i passed), the system transitions to the ReduceDistanceState. This change ensures that the trucks begin to close the gap between them, returning to their optimal formation. This step is crucial for regaining fuel efficiency and maintaining proper platoon coordination. 6. ReduceDistanceState (Returning to Normal Formation) During this state, the trucks gradually decrease their distance to return to the original platoon formation. This prevents unnecessary gaps and ensures that the fuel-saving benefits of platooning are restored. Once the platoon reaches its original spacing, the system moves back to DrivingState, completing the cycle.

G. Activity Diagram

The activity diagram in figure number 9 represents the step-by-step process our platoon system follows when a car attempts to pass through the truck platoon. The diagram outlines how the system monitors its environment, detects obstacles, and dynamically adjusts the platoon's configuration to ensure smooth traffic interaction.

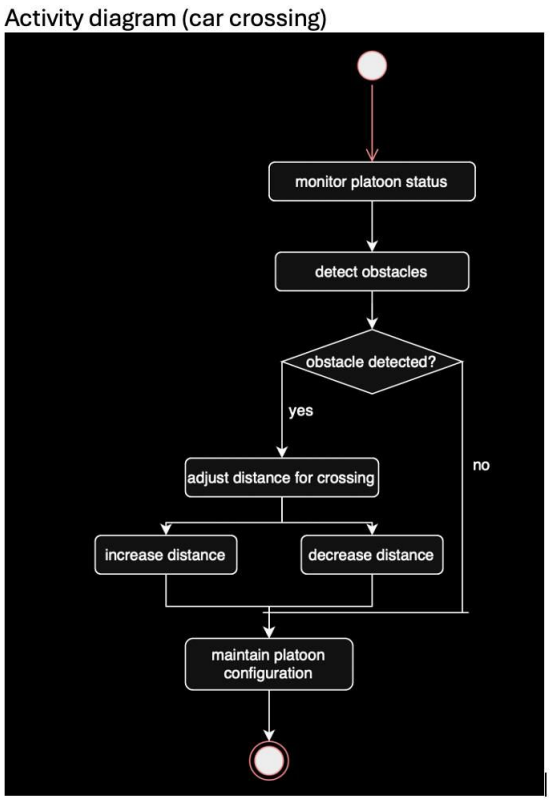

Fig. 9. Activity Diagram

1) Monitoring Platoon Status - The system continuously monitors the platoon's formation and movement to ensure trucks are following the correct distance and speed. This step ensures that trucks remain synchronized and ready to react to any external events.
2) Detecting Obstacles (Recieve message from car) - The system actively scans the surroundings for any obstacles, including approaching vehicles, lane changes, or road hazards. This detection process is crucial to anticipate potential interactions with external vehicles, such as a car trying to merge into the platoon.
3) Decision Point: Is an Obstacle Detected? If no obstacles (e.g., cars, objects, or sudden lane changes) are detected, the system continues normal operation, maintaining the platoon's existing configuration. If an obstacle (such as a car attempting to pass) is detected, the system adjusts

the truck distances to accommodate the passing vehicle safely.

4) Adjusting Distance for Crossing - Once the system confirms that a car is entering the platoon's space, it decides whether the trucks need to increase or decrease distance to allow for a smooth transition.
5) Increasing or Decreasing Distance - If a car is merging into the platoon, the system instructs trucks to increase their distance to make space. If the car has already passed through, the system may instruct the trucks to decrease the distance to restore the platoon's optimal formation. These adjustments happen in real-time to ensure smooth and safe traffic flow.
6) Maintaining Platoon Configuration - Once the car has either successfully passed or merged into the platoon, the system re-establishes the original truck formation. The platoon resumes normal coordinated driving, optimizing for fuel efficiency and traffic safety.

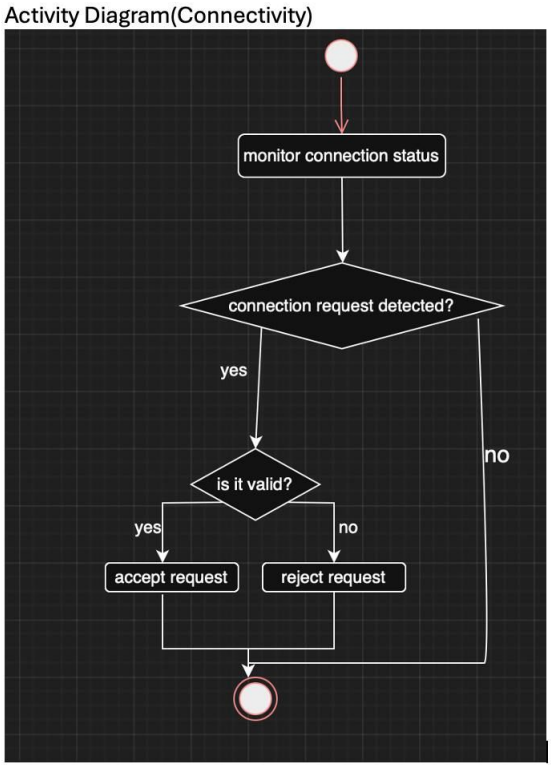

Fig. 10. Activity Diagram(Connectivity)

This activity diagram in figure number 10 represents how our system handles connection requests in a structured and secure manner (like when a car wants to pass or even connection between trucks to increase or decrease distance). The process ensures that only valid connection requests are accepted, maintaining reliable and safe communication between trucks in the platoon.

1) Monitoring Connection Status - The system continuously monitors the connection status to check if any new trucks or external entities are attempting to establish communication. This ensures that all communication requests are processed in real-time.
2) Checking for Connection Requests - When a connection request is detected, the system evaluates whether it is a legitimate request from a recognized source (e.g., a new truck joining the platoon or a necessary external entity like car). If no connection request is detected, the system loops back and continues monitoring.
3) Validating the Connection Request - If a connection request is detected, the system checks whether the request is valid. This validation step ensures that unauthorized or corrupted requests are not accepted, maintaining the security and integrity of the platoon's communication.
4) Accepting or Rejecting the Request - If the request is valid, the system accepts the request, allowing the new truck or entity to establish communication with the platoon. If the request is invalid, it is rejected, and the system returns to monitoring connection status. This process prevents unauthorized access, fake connection attempts, or communication failures that could disrupt the platoon's operation.

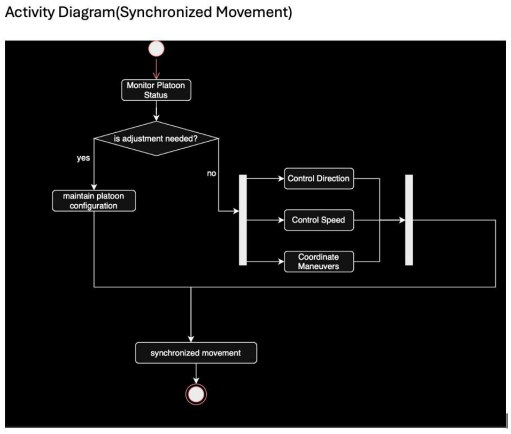

Fig. 11. Activity Diagram(Synchronized Movement)

This activity diagram in figure 11 illustrates how our platoon system ensures synchronized movement by continuously monitoring and adjusting the trucks' configuration. The system dynamically controls direction, speed, and maneuvers to maintain a stable and efficient platoon.

1) Monitoring Platoon Status - The system constantly monitors the platoon's status, checking for any changes in road conditions, truck behavior, or external obstacles. This ensures that the trucks remain aligned and function smoothly as a unit.
2) Decision: Is Adjustment Needed? The system evaluates if any modifications are necessary for the platoon's movement. If no adjustment is needed, the trucks continue to move in a synchronized manner without changes. If an adjustment is required (e.g., due to a lane change, external vehicle interaction, or speed fluctuation), the system proceeds with correction mechanisms.

3) Adjusting the Platoon (If Needed) - If movement adjustments are necessary, the system performs three key control actions in parallel to ensure smooth operation: Control Direction → Adjusting the steering of individual trucks to maintain lane positioning. Control Speed → Managing acceleration or deceleration to align with the platoon leader and traffic conditions. Coordinate Maneuvers → Ensuring that any required platoon movements (e.g., merging, lane changes) are properly executed. These operations happen simultaneously, ensuring real-time synchronization.
4) Returning to Synchronized Movement - After any necessary adjustments, the system ensures that the platoon resumes normal formation and stability. The trucks continue moving in perfect coordination, maintaining efficiency and safety.

H. Activity Diagram

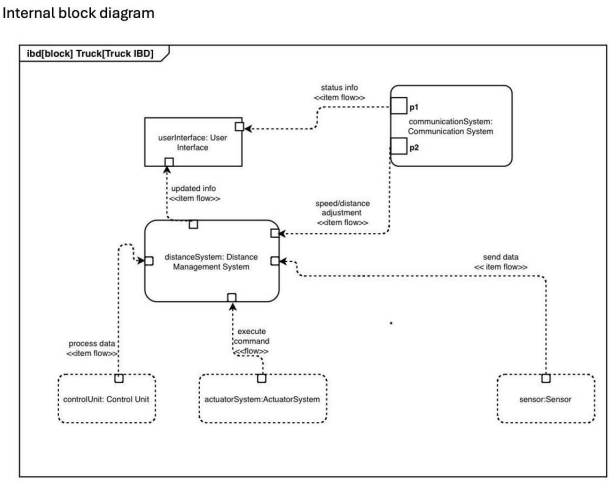

Fig. 12. Internal block diagram

This Internal Block Diagram (IBD) in figure 12 represents the internal structure of our Truck Platooning System, showing how different components interact to ensure efficient distance management, communication, and control within the platoon. The diagram outlines how data flows between key subsystems, ensuring smooth operation and coordinated movement.

1) Distance Management System The Distance Management System is the central unit responsible for maintaining proper spacing between trucks in the platoon. It processes sensor data, executes control commands, and communicates with other components to ensure smooth and safe platooning.
2) Communication System The Communication System exchanges status information with the Distance Management System. This ensures that trucks remain aware of each other's position, speed, and braking actions, enabling synchronized movement.
3) User Interface The User Interface provides updated information to the driver or control system. It receives real-time data from the Distance Management System, helping monitor platoon status and make adjustments when needed.
4) Sensors The sensor system continuously collects real-world data, such as: Distance to the next vehicle,Speed adjustments,External obstacles. This information is sent to the Distance Management System, where it is processed to determine necessary actions.
5) Control Unit The Control Unit processes incoming data from the Distance Management System. It makes key decisions regarding speed adjustments, braking, and acceleration to maintain smooth platoon movement.
6) Actuator System The Actuator System executes commands from the Control Unit by adjusting: Throttle System → To control acceleration. Braking System → To regulate speed and stopping distance. This ensures that the truck responds efficiently to changes in distance, speed, or obstacles.

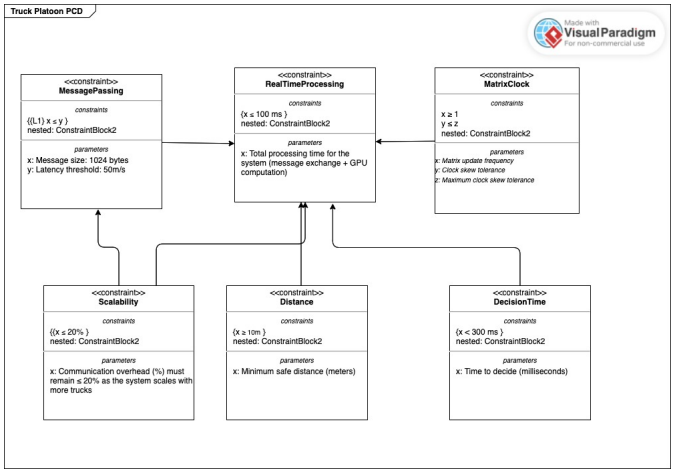

Fig. 13. Parametric Constraint Diagram

The Parametric Constraint Diagram (PCD) as seen in figure 13 defines key constraints and performance limits for the truck platooning system, ensuring efficient and safe operation. These constraints govern message passing, real-time processing, synchronization, scalability, safety distance, and decision-making.

- Message Passing Constraint: Ensures that message exchange between the leader and follower trucks adheres to latency and message size limits. It ensures reliable and fast communication between platoon members to maintain coordination.
- Real-Time Processing Constraint: Ensures that the total system processing time remains within an acceptable limit. This guarantees that platoon reactions to external conditions occur in real time without excessive delays.
- Matrix Clock Constraint: Defines the synchronization constraints for clock updates in the system. It ensures accurate synchronization among platoon members to prevent timing discrepancies.
- Scalability Constraint: Limits communication overhead to maintain system efficiency as more trucks are added to

the platoon. It prevents excessive network congestion as the number of trucks in the platoon increases.

- Distance Constraint: Defines the minimum safe distance between trucks in the platoon. According to the diagram in figure 13, the trucks must maintain a minimum safe distance of 10 meters.
- Decision Time Constraint: Ensures that the system makes decisions (such as adjusting speed or allowing cars to pass) within a strict time limit. According to the diagram in figure 13, reaction time must be less than 300 milliseconds.

IV. Implementation

In this section of the documentation, the paper details the implementation part that has been done. We have chosen to implement some parts of the whole Truck Platooning system. The section will begin with the simulation of the communication part of the system. The details will include, how we realize socket programming, which protocol have been chosen and why. Then, the paper goes in depth into explaining the GPU implementation and the logical matrix clock. Furthermore, this section includes the implementation of the distance control manager and the details on how we realize using the environment of Tinkercad. Lastly, a section is dedicated for scheduling analysis where we identified some tasks and the scheduling analysis for the system.

V. Communication System

For this part of the implementation, our goal is to simulate the communication elements between three different distributed entities or processes. The use case that we have chosen for the simulation is based on the described use case diagrams mentioned earlier in the paper specifically, the car passing scenario. The three processes are Leader Truck, Follower Truck and Car. The main distributed model that we have chosen was based on a Server-client model. This is because we have chosen the protocol of TCP/IP. The main reason why we chose TCP/IP is we can ensure that the message is received by the client at least once when compared to other protocols like UDP where the message is broadcasted and does not ensure the receiver or client receives the message. Another advantage is that with this protocol, is that it is suitable for prototype development. With this fact we can ensure predictability for our Embedded system.

Next, we will move into the details of the implementation. Firstly, we identified the type of message that is to be send by the different entities like Car, Leader and Follower Truck. We achieved this by designing a class diagram for the message. The class will act as a template where, when we create the object message, the details must be filled. Some details of the attributes in message class includes, the string message, id number, timestamp of the message sent, receiver port, validity check for the message and a matrix clock. The operations of the class include serialize() to serialize the message object into a suitable format before sending, and deserialize() to re-create the object in the receiver side. The details can be seen in the figure 14.

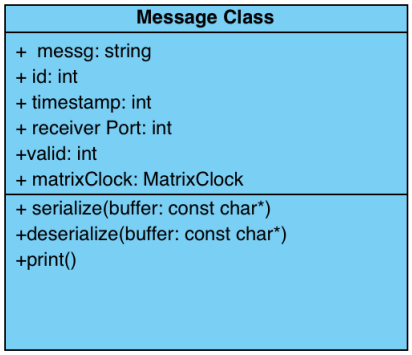

Fig. 14. Communication System

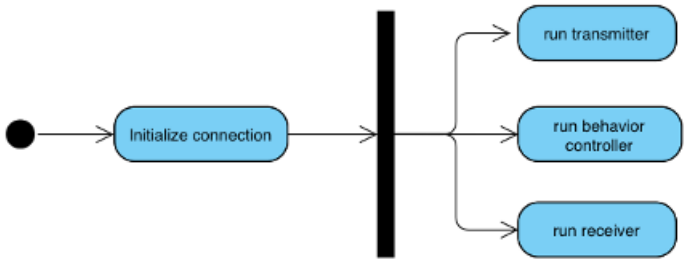

Fig. 15. Communication protocol 1

After that, we identified and implemented the three main tasks that will run in parallel for each of the entities. The tasks are based on the main components described in the node specific architecture. There are three tasks which are transmitter, receiver and a behavior controller. The tasks are implemented as threads and will run once a connection between server and client is initialized. This is described in figure 15. For the receiver module, first it will loop forever and when it receives a message, the message object will be deserialized and stored the object into a message receiver queue. This can be visualized by figure 16.

Next is the transmitter module, it will constantly read from the transmitter message queue and if there is an object to be sent, the object will be serialized and send it. This can be visualized figure 17. Lastly, for each entity there will be a specific behavior module. For simplicity of explanation, we have generalized the module into what it does. First it will read the receiver message queue and if it is not empty, the message will be processed and stored in the transmitter queue and from there on the transmitter module will take over. This is described in figure 18. For the leader truck there will be two ports. One is for V2X communication to handle the messages between leader and car, the other will be for

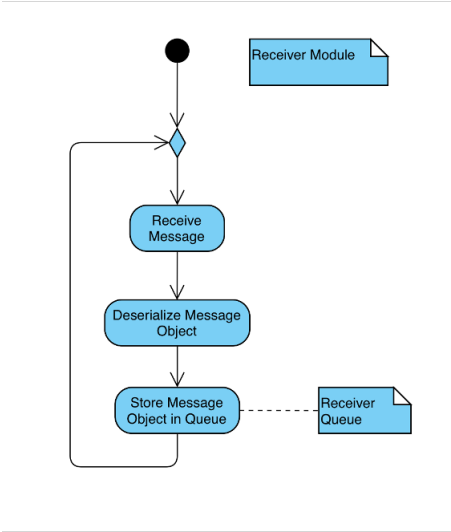

Fig. 16. Communication protocol 2

V2P communication to handle between leader and platoon. Therefore, for the leader truck, there will be four different queues which are V2X receiver and transmitter queues, and V2P receiver and transmitter queues. This design choice is based on the reason that we want a security feature for our system where not every entity can access the leader to platoon communication.

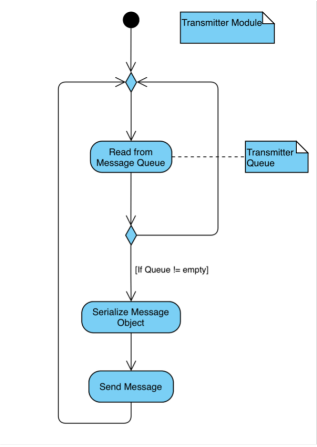

Fig. 17. Communication protocol 3

The output of the simulation can be visualized in figure 19. First we see that server is set up on two different ports. First the car will send a message to the leader that it wants to pass the platoon. Second, the leader will process the message and send the command of increasing distance to the follower truck. Then, the follower will internally increase distance and send a reply to the leader that the distance have been increased. Lastly, the leader will send a message to the car to notify that it is safe to pass. Additionally, we included checks to see whether the messages are valid or not as seen by the figures.

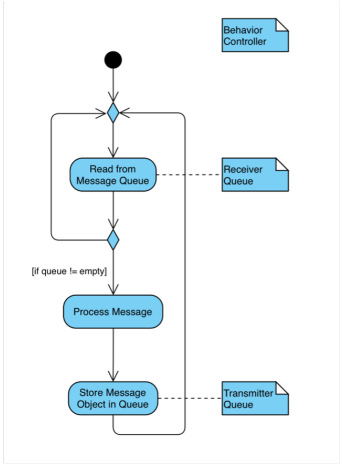

Fig. 18. Communication protocol 4

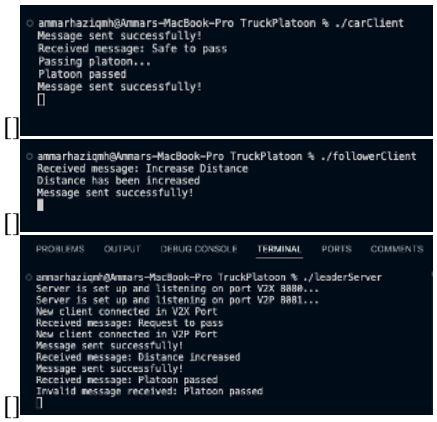

Fig. 19. Communication Output

A. Logical Matrix

A Logical Matrix Clock is an extension of the Vector Clock used in distributed systems to track causality among multiple processes. Instead of maintaining a single vector per process, each process keeps an N×N matrix, where rows represent local knowledge of timestamps, and columns store received timestamps from other processes. This allows a process to not only track its direct interactions but also infer indirect causal relationships, making it more accurate than vector clocks in complex distributed environments. We use Matrix Clocks instead of Vector Clocks because they provide a global view of the entire system's event history, improving causality tracking and reducing false concurrency assumptions. In our project, we used GPU for parallel computations.

Figure 20 is for running our computation on GPU and to check if there is a Violation, change the Result Matrix to 1.

Another part of our platoon system where we can utilize GPU computation is continuously monitoring the distance between the trucks. We have two matrices:

1) Distance Matrix – Represents the distances between the trucks.

```
// CUDA kernel to check distances
__global__ void checkDistances(float *distanceMatrix, bool *violations, int n, float threshold) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < n && col < n) {
        if (row == col) {
            violations[row * n + col] = false;  // Ignore diagonal
        } else {
            violations[row * n + col] = (distanceMatrix[row * n + col] < threshold);
        }
    }
}
```

Fig. 20.

```
Distance Matrix (in meters):
0 15 40 60 80
15 0 10 40 60
40 10 0 18 40
60 40 18 0 25
80 60 40 25 0

Violations Matrix:
0 1 0 0 0
1 0 1 0 0
0 1 0 1 0
0 0 1 0 0
0 0 0 0 0

Kernel Execution Time: 0.173152 ms
```

Fig. 21. Matrix Clock Output 2

2) Violation Matrix – Stores whether a violation has occurred.

The IF statement checks the distance between two trucks in the Distance Matrix. If the row and column indices are the same, it means the truck is being compared with itself, so the distance is 0, and no violation is recorded in the Violation Matrix.

The second IF statement checks whether the distance is less than 20 meters. If it is, this indicates a violation, and the corresponding entry in the Violation Matrix is set to 1.

Finally, we can observe the execution time. When executing the computation multiple times using GPU acceleration, the execution time decreases, demonstrating the efficiency gains from GPU parallelization.

Figure 21 shows our results when we run our program on GPU. As you can see, the Execution time decreased in the Second time with is one of the benefits of GPU Computation.

B. GPU Implementation

The reason why GPUs are so powerful is that they can run thousands of processes simultaneously, whereas normal CPUs do one at a time, hence way faster for deep learning, video processing, and scientific simulations. Whereas CPUs process one task after another sequentially, GPUs operate in parallel, churning through massive volumes of data in record time. If one uses Colab, he might enjoy free GPU by going Runtime ¿ Change runtime type ¿ GPU and checking it works with a command like "!nvidia-smi" . Thanks to libraries like TensorFlow, PyTorch, or CuPy, moving your computation from CPU to GPU will require at most a few

```
Distance Matrix (in meters):
0 15 40 60 80
15 0 10 40 60
40 10 0 18 40
60 40 18 0 25
80 60 40 25 0

Violations Matrix:
0 1 0 0 0
1 0 1 0 0
0 1 0 1 0
0 0 1 0 0
0 0 0 0 0

Kernel Execution Time: 124.59 ms
```

Fig. 22. Matrix Clock Output

lines of code change. For example, it can be seen with the multiplication of large matrices, when several seconds of real time on a CPU are transformed to milliseconds on a GPU. Running the experiment many times makes obvious the heavy superiority of a GPU for such tasks. The aim of the Truck Platooning Project is to deploy GPU acceleration, so real-time message processing among vehicles will be afforded ultra-fast communication and decision-making capability. As the coordination within a platoon involves very fast changes in speed and distance, the application of GPU-accelerated systems would increase the processing rate of sensor data and message exchange to minimize latency in the system.

C. Tinkercad Simulation

The Tinkercad simulation serves as a critical validation tool in the development of the truck platooning system. It allows for the design, testing, and refinement of both hardware and software components before physical implementation. The virtual prototyping of the system using Tinkercad enables the creation of a realistic digital prototype without needing physical hardware, reducing costs and risks. It alos tests the system logic by Simulating sensor inputs, motor behavior, and message passing between trucks to validate control algorithms. It helps with real-time debugging and safe experimentation. For our simulation, we used the following hardwares:

- Arduino Uno (for processing and communication).
- DC Motor (to simulate truck movement).
- Motor Driver (L298N) (to control motor speed).
- Ultrasonic Sensor (to detect an external car).
- External 9V Battery (to power the motor).
- Breadboard and Connecting Wires (for circuit connections).

As seen in figure 23, the leader truck continuously monitors for external cars using the ultrasonic sensor. Upon detection, the leader truck sends a message (signal) to the follower truck via Arduino communication. The follower truck reduces its speed, creating a safe distance. After the car has passed successfully and the sensor no longer detects any external car at close range, the platoon goes back to normal mode. At

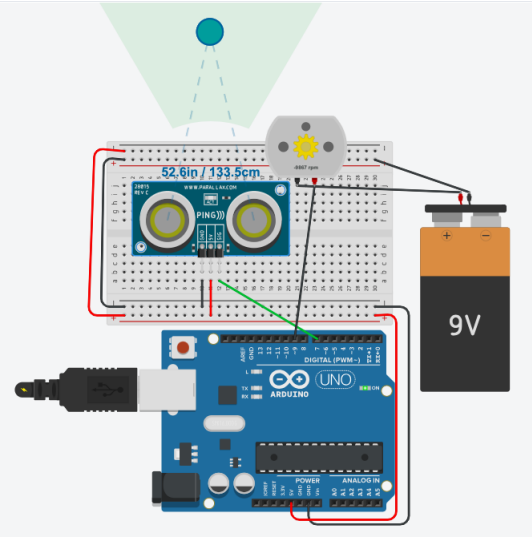

Fig. 23. Hardware Simulation

Normal mode, both leading truck and follower trucks move at the same speed, maintaining a fixed close distance.

```
Distance: 52 in | 133 cm
Distance: 52 in | 133 cm
Distance: 20 in | 53 cm
Increasing distance...
Distance: 17 in | 44 cm
Distance: 17 in | 44 cm
Distance: 36 in | 92 cm
Normal Distance...
Distance: 40 in | 104 cm
Distance: 50 in | 127 cm
```

Fig. 24. Simulation Output

Figure 24 illustrates the output of the simulation. Initially, no obstacle was detected within the sensor's range at a distance of 52 inches (133 cm). As the system detected a car at 20 inches, the truck increased its following distance. The obstacle remained close at 17 inches, prompting further adjustments. Once the detected distance increased to 36 inches, the platoon resumed its normal distance.

D. Scheduling analysis

In this sub-section, we go into detailing the scheduling analysis that were made during implementation phase. Firstly, we setup the environment by using an emulation of Arduino MEGA which has one core. Then we identified three different tasks that are included in our system. The first task is Sensor Handler, where we simulate the tasks of reading sensor values and storing the data collected in memory. Next task is the control unit, where we simulate reading the memory, perform computations, then storing to memory. Last task that we identified is actuator handler, where we simulate reading from memory and perform the actions. The second step is we identified the scheduling algorithm that we wanted to use. For our system, we decided to use Rate Monotonic Scheduling (RMS) algorithm. The main reason is that it is simpler to implement due to the static nature of the algorithm. The static nature refers to the allocation of priority for each task during the initial phase. The deadline and period for this algorithm is also the same which helps the simplicity of implementation. Then, we calculated the worst-case execution time for each of the task. This has been achieved by using software where we performed the task for 1000 iteration cycles and calculate the average execution time for each task. Lastly, we perform the analysis based on the set parameters that we have chosen. For example, we set the period and deadline to be 300 microseconds. We proceeded to calculate the utilization to see if the task set is schedulable for RMS. The results can be seen in figure 25. As observed in the figure, the sensor handler has execution time of 129 microseconds thus the utilization is 0.43, control unit has utilization of 0.19 and actuator handler has utilization of 0.113. When we add them all together it is less than 0.779 which means that the task set is schedulable for RMS algorithm.

Tasks	Execution Time (µs)	Deadline (µs)	Period (µs)	Utilization
Sensor Handler	129	300	300	0.43
Control Unit	57	300	300	0.19
Actuator Handler	34	300	300	0.113

Total Utilization = 0.43+0.19+0.113 = 0.733 < B(3) = 0.779
Taskset is schedulable for RMS

Fig. 25. Scheduling Analysis

VI. Verification and Validation

In this section, the paper describes the detailed explanation on what was done for the verification and validation part for the truck platooning system. The section will begin with the first part which describes some verification techniques that were used. Then, we will go through with the validation techniques that was used during the semester project.

A. Verification

Verification refers to the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. [IEEE Std 610.12-1990] In other words, whether we have implemented the functions that we wanted in our system works as we intended to. Therefore, the technique that we used for the development phase of our project was static analysis techniques specifically, code walkthroughs. We did a peer review where the author of the code walks through the code explaining line by line what each piece of code should do. After the development phase, we needed a way to proof for each function that we wanted to implement works as described in the walkthrough. Therefore, we have developed some tests where we isolate the specific functions or

modules implemented in the system and performed said tests. In figure 26 describes all the tests that were done including the description and the results of each test respectively.

Test	Description	Results
Serialize test	Test ensures the serialization logic correctly stores the objects data in memory	successful
Deserialize test	Test that ensures that deserialized data matches the original object	successful
Transmitter test	Test ensures transmitter correctly sends a serialize message through socket	successful
Receiver test	Test ensures receiver correctly receives a message	successful
Car test	Test the car behavior correctly	successful
Leader test	Test the leader truck behavior correctly	successful
Follower test	Test the follower truck behavior correctly	successful

Fig. 26. Verfication Result

The first pair of tests that were done were focused on serializing and deserializing the message object. As described in the previous sections, the message is an object containing multiple data like string of message, timestamp, validity and so on. Therefore, we need to serialize the object into a suitable format for sending and deserializing it back into an object when receiving the object through socket programming. That is the main reason on why testing was done for these sets of specific functions. The next set of tests that were done was primarily focused on the transmitter and receiver modules. This is to ensure that they correctly read a message from the queue and perform the sending and receiving part of the communication process. Lastly, three separate tests were done to check whether the behavior controller for each entity, for example car, leader truck and follower, perform the control logic correctly. As observed in the figure 26, all the tests that were done were all successful indicating that we have properly verified that the system is behaving as what we intended during the initial phase of the development process.

B. Validation

Validation refers to the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements [IEEE Std 610.12-1990]. In other words, we needed to check whether the simulation of the system that we built can be traceable to the requirements. The primary technique that was used by our group was by inspection of all the documents that was done by each team member by another team member. For example, the requirement diagram was made by Adijat Ajoke and was inspected by Ammar Haziq. Following the inspection process some comments are also made by the inspector. The figure **??** shows the summary of all the documents that was made by whom and checked by whom and also the comments for each respectively.

Document	Made by	Inspected by	Comments
Requirements diagram	Adijat Ajoke	Ammar Haziq	Initial diagram did not capture our whole requirements that we want in our system. Changes were made to add more requirements that better fit our system
Use case diagram	Amir Hossein	Ammar Haziq	Initial diagram was already on a good level. Additional details were added regarding the relationship include and extend
Sequence diagram	Ammar Haziq	Adijat Ajoke	Initial diagram was good except for some minor changes in the terms used so it could conform with the rest of the diagrams
Block diagram	Ammar Haziq	Amir Hossein	The diagram effectively illustrates the modular structure and interconnections of the Truck Platooning System, but it lacks details on dynamic communication flows and interactions between components.
Parametric constraint diagram	Adijat Ajoke	Zahra Mahdion	The parametric constraints diagram captures the system's relationships and constraints. Defining the relationships between processing time, message size, would improve the accuracy of the model.
Internal block diagram	Zahra Mahdion	Adijat Ajoke	The internal block diagram provides a clear structural view of the system components and their connections. To enhance clarity, it should explicitly represent the data flow between the sensor module, communication unit, and control unit within both the Leader and Follower Trucks.
State machine diagram	Zahra Mahdion	Amir Hossein	The state machine diagram effectively shows the transitions between states, but it should include the intermediate step where the Leader Truck first receives the message from the car and then communicates with the Follower Truck to increase the distance, ensuring the sequence of actions is accurately represented.
Node architecture	Ammar Haziq	Amir Hossein	The diagram effectively outlines the Node Specific Architecture, but adding labels to the arrows to specify the type of data or messages exchanged between components would improve clarity and understanding.

Fig. 27. Validation Table

VII. Conclusion

This project successfully developed and simulated a truck platooning system by integrating Embedded Systems Engineering (ESE) and Distributed and Parallel Systems (DPS) concepts to model real-world autonomous vehicle coordination. The system maintains an efficient leader-follower truck formation, dynamically adjusting when external vehicles attempt to pass. Utilizing sensors and structured communication protocols, the leader truck detects approaching obstacles and signals the follower truck to reduce speed, creating a safe gap for lane changes. Once the passing vehicle clears the platoon, normal spacing is restored. To enhance performance, the project leverages GPU acceleration for fast message exchange and real-time computation, minimizing latency. Additionally, SysML and UML diagrams were employed to comprehensively model the system architecture, behavior, constraints, and interactions, ensuring a structured and scalable design.

Future improvements could include more advanced sensor fusion, V2X (Vehicle-to-Everything) communication, and machine learning-based decision-making to further optimize platoon coordination. Additionally, deploying the system on physical hardware beyond simulation would provide valuable insights into real-world performance. Overall, this project demonstrates the feasibility and benefits of intelligent truck platooning as a step toward autonomous transportation and smart mobility solutions.

References

[1] Janssen, R., Zwijnenberg, H., Blankers, I. and de Kruijff, J., 2015. Truck platooning: Driving the future of transport. TNO report

[2] Albiński, S., Crainic, T.G. and Minner, S., 2020. The day-before truck platooning planning problem and the value of autonomous driving. CIRRELT.

VIII. APPENDIX

A. Github Repository Structure

The project's GitHub repository is structured to ensure clarity and maintainability. The following structure outlines the key directories and files.

ESE-Group-1:

- **Documentation** – Contains project documentation and affidavit.
- **Homework01** – Contains assigment
- **Implementation** – Holds all source code, including leader and follower truck control logic and communication handling.
- **Modeling Diagrams** – Includes all UML and SysML diagrams, such as sequence, state machine, and parametric constraint diagrams.
- **README.md**

B. Collaboration & Contribution

The success of the truck platooning project was a result of strong collaboration and shared contributions among all team members. The team worked together closely, dividing tasks according to each member's strengths and expertise. Throughout the project, all members were actively involved in the key stages, from planning and system design to implementation and testing.

Each team member contributed in the following ways:

Design and Development: All members participated in the design of the system architecture and developed key components of the platooning system. Simulation and Testing: The team worked together to implement and test the simulation, making sure the platoon's behavior was accurate and responsive to the dynamic traffic conditions. Documentation: The documentation was collaboratively written, with each member contributing to the description of their respective sections, including system architecture, implementation, and test results. Problem Solving: Whenever challenges arose during the development, the team worked together to brainstorm solutions, support each other, and ensure the project continued progressing.