Truck Platooning: A Distributed Peer-to-Peer Approach with TCP/IP Communication

Mohammad Rizwan
Department of Computer Science
Fachhochschule Dortmund
Dortmund, Germany
mohammed.mohammedrizwan009@stud.fh-dortmund.de

Hazhir Amiri
Department of Computer Science
Fachhochschule Dortmund
Dortmund, Germany
hazhir.amiri001@stud.fh-dortmund.de

Anguiga Hermann
Department of Computer Science
Fachhochschule Dortmund
Dortmund, Germany
hermann.anguiga001@stud.fh-dortmund.de

***Abstract*— In this paper, a truck platoon system is developed and implemented based on a distributed and parallel system concept. A Peer-to-Peer architecture is implemented in the system with a communication interface consisting of a server-client socket, enhancing efficiency and communication resilience. In this architecture, each truck, including the leader, prime follower, and followers, operates both as a client and a server, ensuring robust and decentralized decision-making. The system utilizes TCP/IP for reliable communication, allowing for real-time transmission of critical information like speed, direction, and platoon formation. In addition, the system's distributed architecture is discussed along with parallel programming approaches that are also implemented partly in the system. The focus is on leveraging distributed system architectures and parallel programming methodologies, with significant portions of the system being implemented in Java. This approach not only streamlines communication within the platoon but also enhances overall system robustness and scalability. The implementation outcomes, observed through console outputs and the graphical simulation, demonstrate the system's effectiveness in real-world scenarios, offering a promising solution for modern logistics and fleet management challenges.**

Keywords*— *Leader Truck, Prime Follower, Follower Trucks, UML Diagrams, TCP/IP, Peer-to-Peer Architecture

I. INTRODUCTION

Truck platooning is a cutting-edge technology that allows two or more autonomous trucks to drive together in a synchronized manner. Both the leader and the following trucks are equipped with autonomous driving technologies and are interconnected through a computer system, enabling them to communicate and coordinate their movements closely. This technology offers numerous benefits, including improved fuel efficiency, decreased carbon emissions, and more efficient fleet management. The key concept behind truck platooning involves facilitating communication and cooperation among trucks on the road, using a decentralized decision-making system and robust communication channels. By doing so, these platoons can drive in formation, which reduces wind resistance and increases fuel efficiency.

Truck platooning also has other benefits, such as minimizing driver shortages, lowering logistical expenses, and improving driver comfort. The platoon's leader would be the truck in front, with the following trucks adapting to changes in platoon movement that do not require driver involvement. If communication is lost, the following trucks can safely exit the platoon and drive independently to neighboring platooning stations.

To ensure the effectiveness of truck platooning, numerous technologies are used, including Adaptive Cruise Control (ACC) and TCP/IP communication protocols. The use of the TCP/IP protocol over traditional Vehicle-to-Vehicle (V2V) communication systems is a critical component of this study. TCP/IP, a basic communication protocol in the field of network computing, provides strong and dependable communication routes between trucks. TCP/IP, which uses socket addresses for communication, allows the transfer of critical data such as speed, direction, and positioning between trucks in real-time. This assures a high level of accuracy and synchronization, both of which are required for the platoon to operate safely and effectively. TCP/IP provides enhanced security and resilience against potential cyber-attacks, which is a concern with some V2V systems.

Our technique is supported by actual examples, with findings displayed via console outputs and a graphical simulation. These findings demonstrate our system's effectiveness and adaptability in real-world contexts. This research contributes to the ongoing conversation about autonomous vehicle coordination by suggesting an innovative and efficient method for a more sustainable future in transportation.

Overall, truck platooning is an exciting technology that has the potential to reduce fuel consumption, lower emissions, and enhance fleet management as the transportation industry advances.

A. Problem Domain

This paper delves into the analysis, design, and implementation of a sophisticated system based on distributed and parallel computing techniques. It is designed to operate an autonomous truck platoon, which is a convoy of trucks that work in tandem. Each truck, working as an autonomous entity, is outfitted with several processors, which are fundamental to this system. These processors are controlled by a primary server, which is in charge of crucial tasks like coupling and disconnecting trucks within the platoon and performing emergency braking maneuvers when necessary.

Our system's essence is its distributed nature, which includes a network of independent processing units, with each vehicle in the platoon communicating and working together to achieve a common goal. This teamwork is crucial for completing activities that are complex and dynamic characteristics of real-world driving scenarios.

B. Characteristics

In the context of the autonomous truck platooning project, the distributed system's characteristics are critical to assuring the platoon's effectiveness, safety, and efficiency.

1) Robust communication protocols

The system relies on robust communication protocols to facilitate reliable data flow among vehicles. TCP/IP protocols are employed to ensure consistent communication, providing a dependable exchange of key information like speed, direction, and emergency signals. Socket programming enables real-time data sharing among trucks, allowing swift responses to dynamic platoon conditions. Additionally, fail-safe mechanisms are implemented to address communication failures, ensuring platoon integrity. These mechanisms include allowing trucks to operate independently or safely exit the platoon in such situations.

2) Efficient Data Synchronization

The system employs real-time data exchange through synchronization to maintain consistent spacing and coordinated maneuvers within the platoon. Utilizing distributed clocks and synchronization algorithms ensures simultaneous time-sensitive decision-making across the entire platoon.

3) Strong Security Measures

To safeguard our platooning project from cyber threats, all communication lines are encrypted, ensuring data integrity and resistance to unauthorized access. Additionally, the trucks utilize robust authentication protocols, which confirm the identity of each member, maintaining the platoon's security and ensuring that only verified trucks are integrated into the system.

4) Scalability

Our platooning system has a modular architecture that allows for easy integration of additional vehicles without requiring extensive adjustments. This versatility enables the system to easily support a variety of platoon sizes. Furthermore, the system excels in resource allocation, ensuring optimal performance even when platoon size varies, as it dynamically reallocates resources for efficient operation.

5) Fault Tolerance

The platooning system prioritizes dependability through redundancy, adding backup systems for important components to reduce downtime in the event of problems. Furthermore, the device features self-diagnostic capabilities that enable the detection of faults in certain trucks. In response to such obstacles, the system automatically restructures the platoon, which may involve rearranging the formation or allowing a truck to leave for repairs, to ensure continuous and efficient operation.

6) Flexibility and Decentralization

The platooning system is smart – it can adapt to different situations like traffic or road changes. Each truck makes its own decisions, so if one part has a problem, it doesn't affect everything. This makes the system strong and reliable.

C. Requirements

In many engineering designs, one of the most important steps in modeling the system is to identify and define the system's requirements [1]. Here, we will follow the same approach. The primary requirements in designing this truck platooning system include advanced communication infrastructure, reliable sensor technologies, and standardized protocols. The interconnected communication infrastructure is crucial to enable real-time data exchange among platooning trucks, allowing them to synchronize their movements closely. This requires robust and low-latency communication systems to facilitate seamless coordination. As discussed, we will use the TCP/IP protocol suite. Additionally, the integration of reliable sensor technologies is essential to ensure an accurate perception of the surrounding environment. These sensors enable the trucks to maintain optimal spacing and respond swiftly to changes in traffic conditions.

Fig. 1 depicts some requirements in a requirement diagram. Defining these requirements is vital for project success, stakeholder involvement, measurability, traceability, prioritization, consistency, thorough documentation, and validation and verification processes.

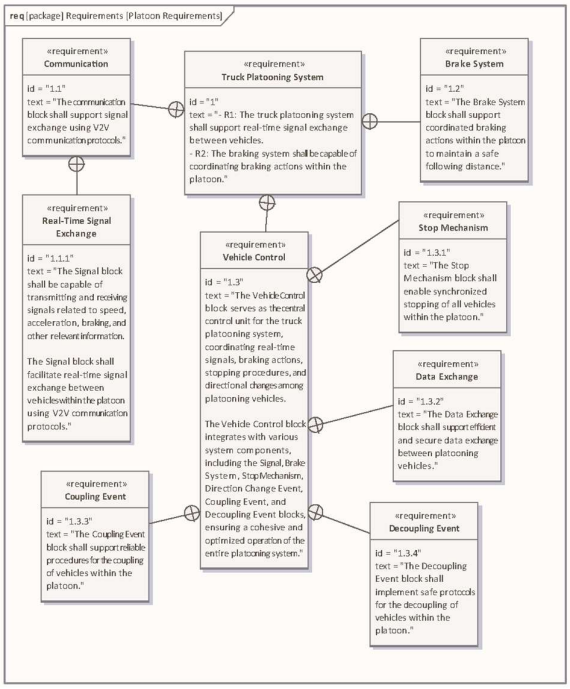


Fig. 1. Platoon requirements diagram

II. PLATOON MODEL

A. Network Model

The network structure resembles a Peer-to-Peer communication model [2], where each node (truck) can act both as a client and a server. This means that while the leader truck coordinates the overall journey, individual trucks can directly communicate with each other without always going through the leader. They can ask each other to maintain the distance between them. The Leader will coordinate how the platoon should be formed with the available trucks and selects the vice-leader (known as the Prime Follower) who

acts as a leader when the Leader loses communication with the platoon. If the Leader loses the communication, the Prime Follower takes control of all the communication and becomes the Leader, and updates the platoon, choosing a new vice-leader in the process. This scenario can go on until the platoon is completely disconnected.

This approach enhances the system's resilience and flexibility, as trucks can share information, manage connections, and even reconfigure the platoon without relying solely on the Leader. The Peer-to-Peer aspect is especially useful for maintaining communication and operation if the leader truck encounters issues or goes out of service.

The entire network model follows mesh topology and message-passing as the mode of interaction.

B. Mesh Topology & Message Passing

In our autonomous truck platooning project, we integrate a mesh network topology to provide a robust and flexible communication framework [3]. In this setup, each truck is capable of communicating directly with any other truck in the platoon, not just its immediate neighbors. This aspect of the topology is crucial for redundancy, ensuring that if the linear chain is disrupted (e.g., a truck leaves the platoon), the system can reroute communications through other trucks. It also facilitates direct peer-to-peer interactions for specific use cases like joining or leaving the platoon, emergency broadcasts, or sharing sensor data.

Each truck in the platoon functions as a network node, able to create direct TCP/IP socket connections with its counterparts. This dual capability for initiating and receiving connections is critical in handling the dynamic nature of truck platooning, which allows vehicles to effectively join or leave the formation.

The use of a mesh topology provides several significant advantages to our system. For starters, it improves reliability; with numerous channels available for data transmission, the network may reroute communications if a truck loses connectivity, ensuring continuous information flow. Second, the system's scalability is clear, as the mesh network can easily accommodate the addition or removal of vehicles without requiring large reconfigurations. Third, the decentralized design of a mesh network eliminates a single point of failure, with one vehicle capable of communicating with multiple others, dispersing network load and lowering the danger of collapse. Furthermore, direct truck-to-truck communication promotes faster response times and real-time data sharing, which are critical for ensuring safe platooning operations. Finally, the efficient data distribution feature of a mesh network is ideal for broadcasting information to all trucks in the platoon.

In essence, the mesh topology serves as the foundation for our autonomous truck platooning system. It not only assures dependable and scalable communication, but it also considerably improves the overall safety and operational efficiency of platooning trucks on the road.

C. TCP/IP Protocol

The TCP/IP protocol is used as a fundamental communication backbone in our truck platooning project, which is built on a peer-to-peer architecture. This integration creates seamless and direct communication channels between individual vehicles in the platoon, resulting in a decentralized approach to information exchange. It assures that the data sent is error-free, full, and in the sequence in which it was sent. IP is in charge of data routing as well as data integrity. It allows heterogeneous networks to communicate across platforms. It is a client-server architecture that is scalable. This enables new networks to be introduced without interfering with existing services.

Each vehicle in the platoon functions as an autonomous node, able to make direct TCP/IP connections with other trucks. This peer-to-peer paradigm eliminates the need for a central server, resulting in a more robust and fault-tolerant system. The TCP component of the protocol ensures that data packets are transmitted reliably, with order and integrity. This is especially important for the interchange of critical information like speed, distance, and navigation data, where precision is essential.

The peer-to-peer architecture using TCP/IP easily accommodates the dynamic nature of truck platooning, which is characterized by trucks joining or departing the formation. The network is dynamically reconfigured, allowing vehicles to create new connections or terminate current ones while responding to changing platoon configurations and maintaining overall connectivity.

TCP/IP's intrinsic ability to manage changeable network conditions such as packet loss, shifting signal intensities, and latency makes it ideal for the mobile context of truck platooning. Furthermore, the peer-to-peer model's decentralized control and coordination allows each truck to share information directly with its peers, strengthening the system's resilience in the face of individual unit failure.

The scalability and flexibility inherent in the peer-to-peer architecture, facilitated by TCP/IP, are critical for handling various platoon sizes and configurations. Furthermore, the solution employs TCP/IP for additional security measures, allowing for powerful encryption and authentication procedures at the individual truck level to assure the security of data exchange inside the platoon.

D. UML Digrams

A UML diagram is a diagram based on the UML (Unified Modelling Language) which is used to visually represent a system, such as its key figures, roles, actions, artifacts, or classes, in order to better understand, alter, maintain, or document system information.

1) State machine diagram

A state machine diagram illustrates the behavior of a single object by describing the sequence of events that an object experiences during its existence as a result of external events.

The state machine diagram in Fig. 2 shows the possible states and transitions of a truck that interacts with a platoon. A platoon is a group of vehicles that travel together in a coordinated manner, using wireless communication and sensors to maintain a constant speed and distance. A vehicle can join or leave a platoon at any time, depending on its preferences and goals.

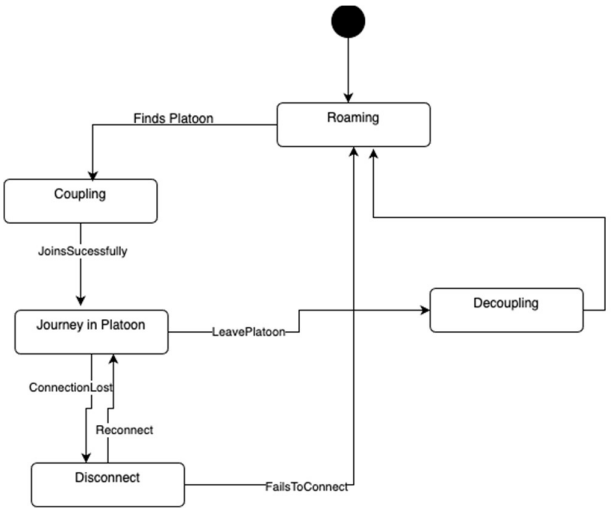


Fig. 2. Truck state machine diagram

The initial state of the vehicle is "Roaming", which means that it is traveling independently without any connection to a platoon. When the vehicle detects a platoon nearby, it enters the "Coupling" state, where it attempts to join the platoon by sending a request and following the platoon leader's instructions. If the joining process is successful, the vehicle transitions to the "Journey in Platoon" state, where it becomes a member of the platoon and follows the platoon's speed and trajectory. During the journey, the vehicle continuously monitors the connection quality with the platoon leader and other members. If the connection is lost, the vehicle tries to reconnect by sending a signal and adjusting its position. If the reconnection is successful, the vehicle resumes the journey in the platoon. If the reconnection fails, the vehicle disconnects from the platoon and returns to the "Roaming" state. Alternatively, the vehicle can also decide to leave the platoon voluntarily by sending a notification and moving away from the platoon. In this case, the vehicle enters the "Decoupling" state, where it completes the leaving process and then goes back to the "Roaming" state.

The state machine diagram captures the essential features and behaviors of a vehicle's interaction with a platoon and provides a clear and concise representation of the system's logic and dynamics. It can be used as a basis for designing and implementing the platooning system.

The state machine diagram in Fig. 3 consists of four primary states: "Follower," "Prime Follower," "Foreigner," and "Leader." A Follower is an entity that follows the Leader's instructions and maintains a safe distance from other entities. A Prime Follower is a special Follower that is elected by the Leader to act as a backup in case of Leader's failure. A Foreigner is an entity that is not initially part of the platoon but wants to join it. A Leader is an entity that controls the platoon's speed, direction, and formation.

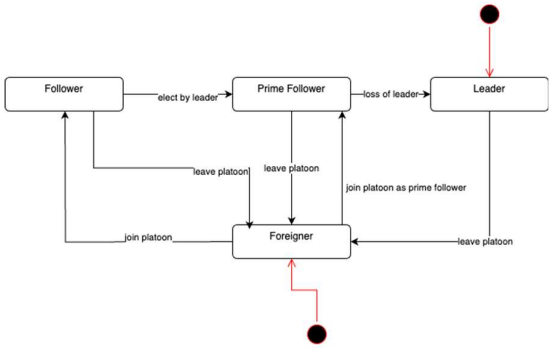


Fig. 3. Roles state machine diagram

The diagram also shows the possible transitions between these states, along with the conditions or actions that trigger them. For example, a Follower can become a Prime Follower if the Leader elects it, or it can leave the platoon and exit the system. A Prime Follower can become a Leader if the current Leader fails, or it can revert to being a Follower and if it leaves the platoon, it becomes Foreigner. A Foreigner can join the platoon as a Prime Follower if the Leader accepts it, or join as a Follower. A Leader may only leave the platoon and become a Foreigner.

The driving control system has two main modes: "human_mode" and "autonomous_mode". The possible SysML state machine diagram for this system is shown in Fig. 4. Human driving and autonomous modes are depicted as basic states. Fig. 5 displays the state machine related to the autonomous mode. The sub-machine state within the autonomous mode includes both an initial and a final pseudo-state. Additionally, it features an orthogonal state with two regions capable of running concurrently. The upper region illustrates the standard autonomous driving behavior during highway mode. The second region within the autonomous mode is dedicated to collision avoidance, which consists of two primary states: active and inactive modes. The inactive mode is represented by a simple state, while the active mode is illustrated as a sub-machine state. Fig. 6 depicts the state machine for the active mode of the collision avoidance state. This sub-machine becomes active when a potential front-end or rear-end collision is detected. Within the active mode of collision avoidance, orthogonal regions enable the concurrent activation of various collision avoidance controls [4].

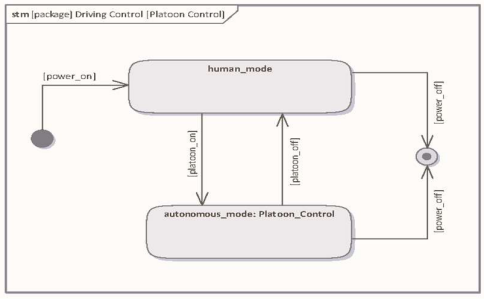


Fig. 4. Platoon control module

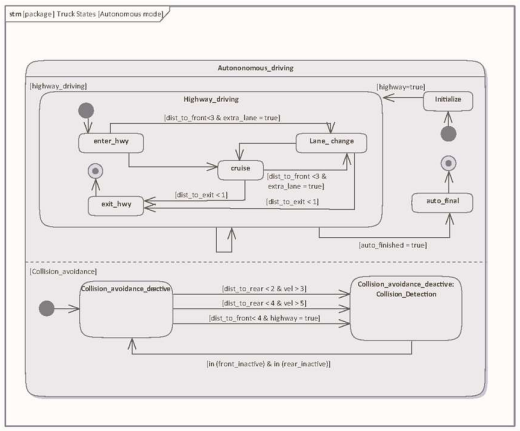


Fig. 5. Autonomous driving mode

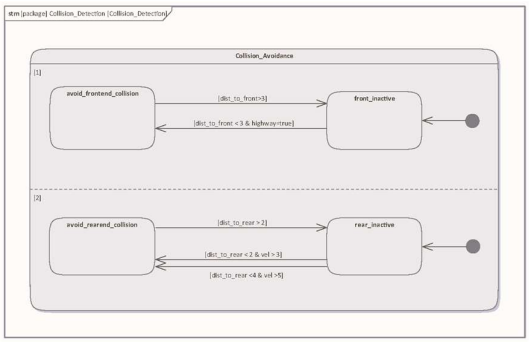


Fig. 6. Collision detection

2) *Activity diagram*

The activity diagram in Fig. 7 shows the protocol followed by a leading truck and a following truck in the event of communication loss or failure during platooning.

The diagram consists of two parallel paths for the leading truck and the following truck, starting from the initial state of "Communication Loss Detected." The leading truck tries to reconnect with the following truck for N times, where N is a predefined parameter. If the reconnection is successful, the leading truck continues platooning with the following truck. If the reconnection fails, the leading truck removes the disconnected follower from the platoon and adjusts the platoon accordingly.

The following truck, upon detecting the communication loss, activates the first following vehicle (FV) as an acting leading vehicle (LV). The FV is the truck that is directly behind the following truck in the platoon. The following truck then tries to reconnect with the LV for N times. If the reconnection is successful, the following truck continues platooning with the LV. If the reconnection fails or if there is no FV available to act as LV, the following truck activates the autonomous drive mode and parks in a nearby parking lot.

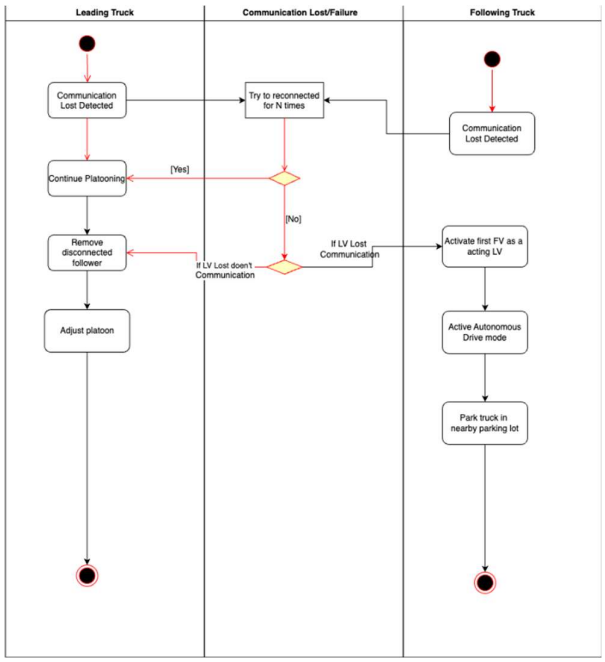


Fig. 7. Communication activity diagram

3) *Sequence diagram*

The sequence diagram in Fig. 8 shows the interaction process among four entities: LeaderTruck, PrimeFollower, FollowerTruck1, and FollowerTruck2 in a platoon driving scenario.

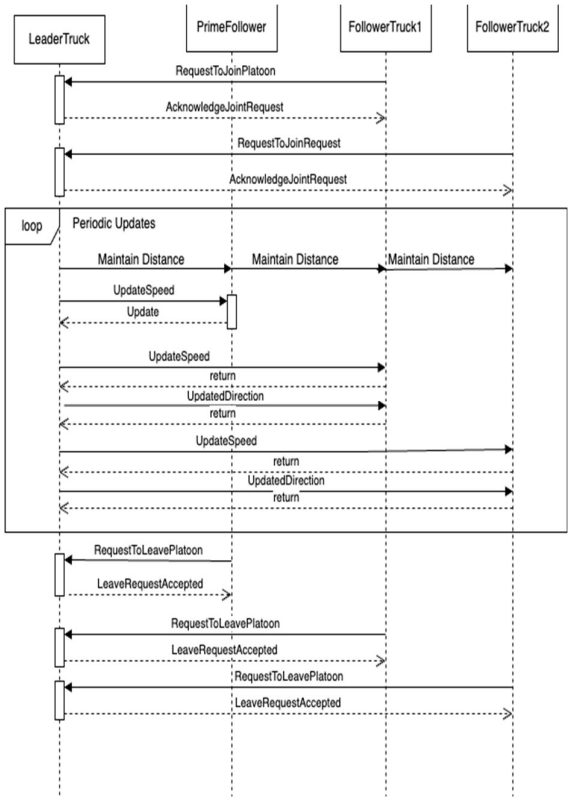


Fig. 8. Communication sequence diagram

At the beginning of the interaction, the trucks request to join the platoon by sending "RequestToJoinPlatoon"

messages to the LeaderTruck. The LeaderTruck acknowledges their requests by sending back "AcknowledgeJoinRequest" messages. Within a loop of periodic updates, each truck is responsible for maintaining a safe distance from the truck in front of it by sending "Maintain Distance" messages. Each truck also updates its speed and direction by sending "UpdateSpeed" messages. The LeaderTruck and the PrimeFollower return updated speed and direction information as responses to these messages.

At the end of the interaction, any truck can request to leave the platoon by sending "RequestToLeavePlatoon" messages to the LeaderTruck. The LeaderTruck accepts their requests by sending back "LeaveRequestAccepted" messages.

The sequence diagram in Fig. 9 represents the platoon leader handing over authority to Prime Follower, which occurs after the LeaderTruck loses communication with the platoon. When communication fails, the LeaderTruck become inactive, and the PrimeFollowerTruck, located right behind it, assumes leadership.

The routine begins with the PrimeFollowerTruck detecting the leader's failure and automatically commencing the handover. It then continuously updates and transmits its speed and direction to the other trucks, adjusting for road conditions. After receiving these inputs, the follower trucks alter their movements to ensure the platoon's steady and safe formation. The method includes an optional interaction in which any vehicle can request to leave the platoon; if authorized, the truck exits formation. The routine successfully culminates with the PrimeFollowerTruck taking up all leadership responsibilities, ensuring that platoon activities continue despite the first leader's failure.

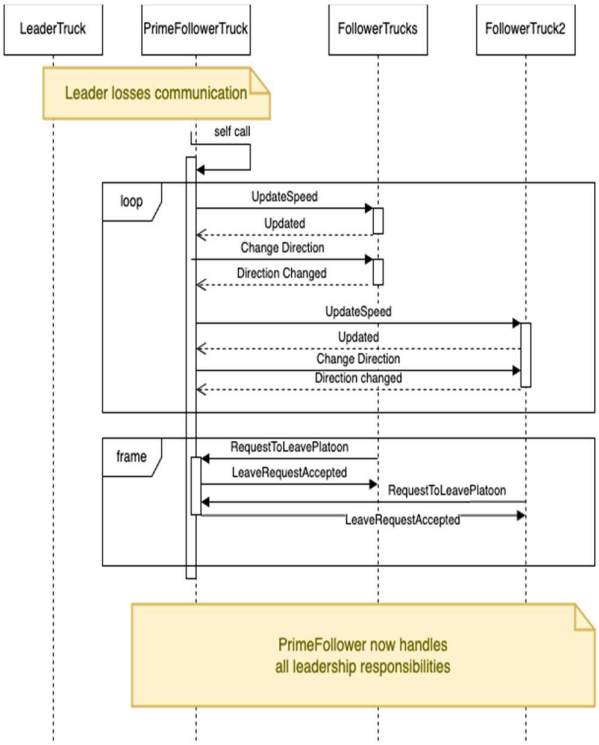


Fig. 9. Communication diagram when Leader loses connection

4) Use case diagram

The use case diagram in Fig. 10 shows that only the leader has the power to start or end the platoon, giving centralized control. On the other hand, each truck can independently join or leave the platoon. Moreover, trucks can automatically keep a safe distance from other vehicles on the road, improving overall safety. They also have communication abilities, allowing them to interact smoothly with other trucks in the platoon. This breakdown of roles and capabilities sets up an organized and effective system for how the platoon operates.

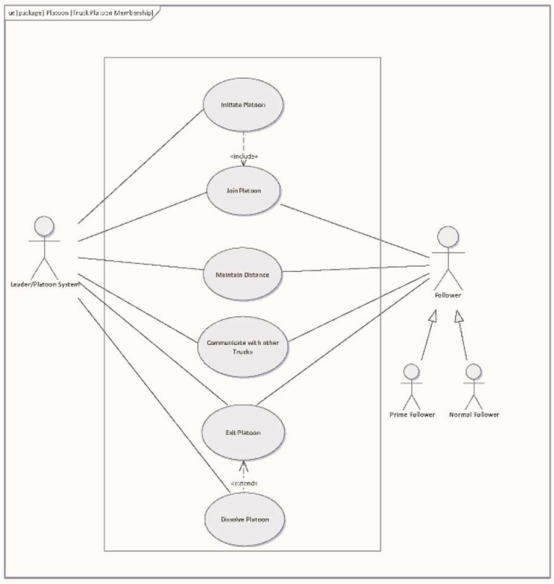


Fig. 10. Truck platoon membership

Hazards frequently present themselves along the roadway, and these can manifest as foreign objects or the presence of other vehicles, impeding traffic flow. Fig. 11 illustrates the mechanisms designed to prevent collisions with such obstacles. These mechanisms involve the activation of sensors, such as distance sensors or motion sensors, which play a crucial role in identifying potential dangers. These sensors are tuned to detect both stationary and moving vehicles. The Collision Detection process comprises a set of functions that interpret the data obtained from these sensors, generating a meaningful output. Subsequently, this output is used in the "Avoid Collision" function, which dictates various actions to be taken by the truck in response to an impending collision. These actions may include altering the vehicle's heading, engaging the brakes, or adjusting the steering to ensure a safe maneuver [5].

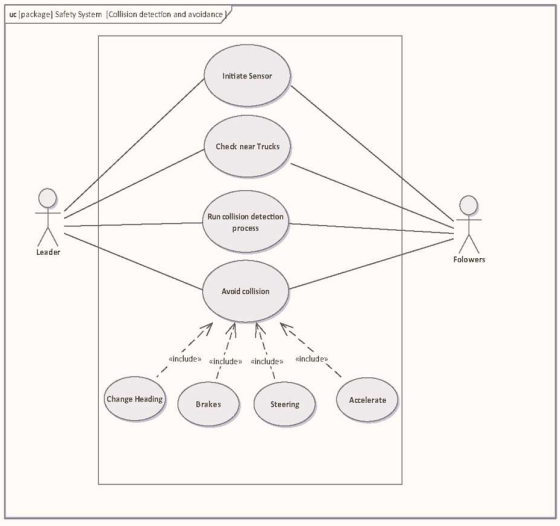


Fig. 11. Collision detection process use case

E. System Architecture

A platoon system is usually made of major and minor modules, also called functions. With all minor modules being sub-sets of major ones. The architecture of our truck platooning system is defined in a SysML block definition diagram (BDD) (see **Error! Reference source not found.**). An important aspect of this model is the number of functions considered. For simplicity, only a few make up the basis of this design.

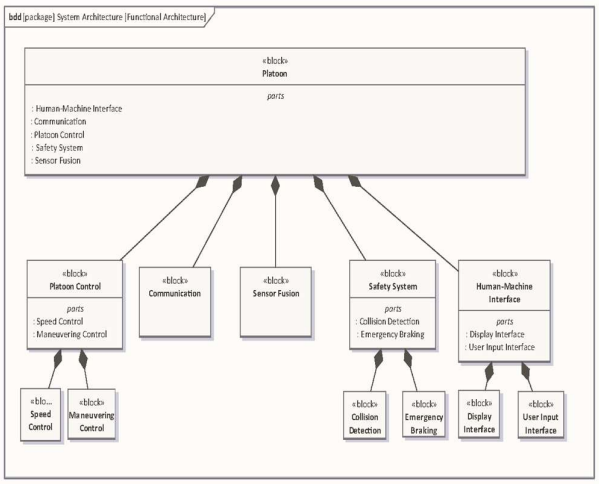


Fig. 12. Platoon functional architecture

In Platoon Control module sub-blocks are located functions related to the motion of the truck. The Speed Control block is the location where the platoon system adjusts the truck's speed dynamically based on real-time traffic conditions and platoon strategies. In the Maneuvering Control block, the platoon system executes safe and coordinated lane changes within the platoon and controls the truck's turning maneuvers, ensuring smooth and synchronized movements. The Communication module is most important for real-time data exchange. This is where the exchange of information on speed, acceleration, and braking with other trucks in real time takes place. Data from the surroundings are collected in the Sensor Fusion module. Different sensors such as distance sensors will, in real-time and in a constant manner, provide trucks with distance information. The Safety System module is the guarantor of destination arrival. It has two sub-modules. Collision Detection and Emergency Braking. The role of the first mentioned (module) is to constantly check on the relative distance of the truck to the rest of the platoon and the overall traffic. The truck must maintain a certain distance called safe distance which is also a function of speed. To avoid a collision, the truck must maintain a safe distance. In the second sub-module are embedded components relative to the braking in an emergency. In situations where collision is detected, this function will be solicited. the last module is The Human-Machine Interface. It is made of two sub-modules. The Display Interface sub-module displays real-time information on platoon status, including following distance and system health, and the User Input Interface sub-module which allows the driver to manually override certain platooning decisions.

Fig. 13 reveals many details on the Sensor Fusion module. Here, the internal components of the module are shown. Control, Fusion Logic, and Radar are the three major ones.

The Control component uses the fused sensor data to make decisions and control the vehicle's behavior in the platoon. The Fusion Logic component contains algorithms and processing units responsible for merging and interpreting data from multiple sensors. The Radar component consists of multiple radar sensors strategically placed on the truck to provide coverage of the surrounding environment. Each radar sensor emits radio waves and detects reflections from nearby objects to estimate their position, velocity, and size. It has a transmitter which is responsible for generating and emitting electromagnetic waves. The transmitted wave travels through space and interacts with objects in its path. Its third component is the Receiver. is responsible for capturing and processing the echoes received by the antenna. It amplifies and filters the received signals to extract relevant information.

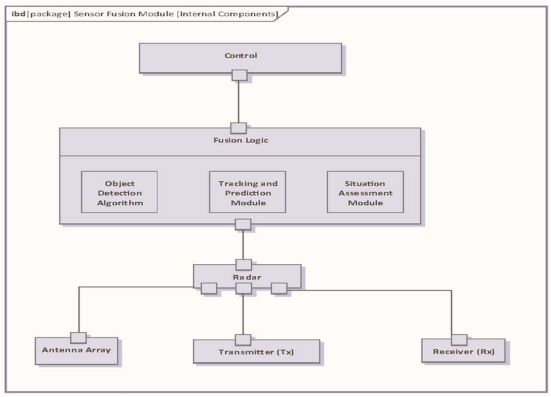


Fig. 13. Fusion Sensor Module internal structure

Another module that deserves our attention is the Human-Machine Interface (HMI) shown in Fig. 14. This is an interface that allows interaction between humans and machines. It is made of many components. The major component Display Interface comprises mainly visual-related components. Visual Elements is its first sub-component. It encompasses the graphical user interface (GUI) elements displayed on the screen, including text, icons, graphics, and animations. It also has the Display Unit sub-component which includes the screen that presents visual information to the driver. Its last sub-component is Information Layout. This

component involves the arrangement and organization of information on the display unit to optimize readability, usability, and accessibility for the driver. Next to the Display Interface is another major component called User Input Interface. This component is made of many parts. The sub-component Input Devices consists of physical or virtual input devices that allow the driver to interact with the platooning system. Input Methods sub-component defines the methods by which the driver inputs commands, adjustments, or selections into the platooning system. It includes interactions such as pressing buttons, rotating knobs, sliding controls, and tapping icons on a touchscreen. Finally, the Feedback Mechanisms sub-component provides feedback to the driver in response to their inputs, confirming actions, acknowledging commands, or providing status updates.

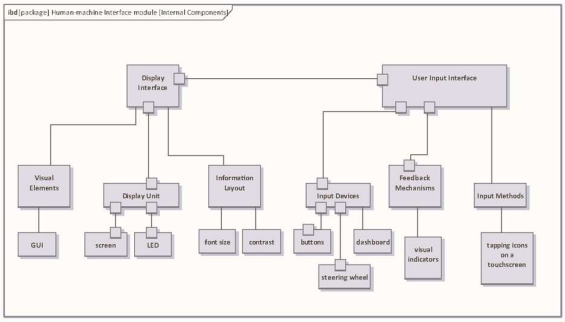
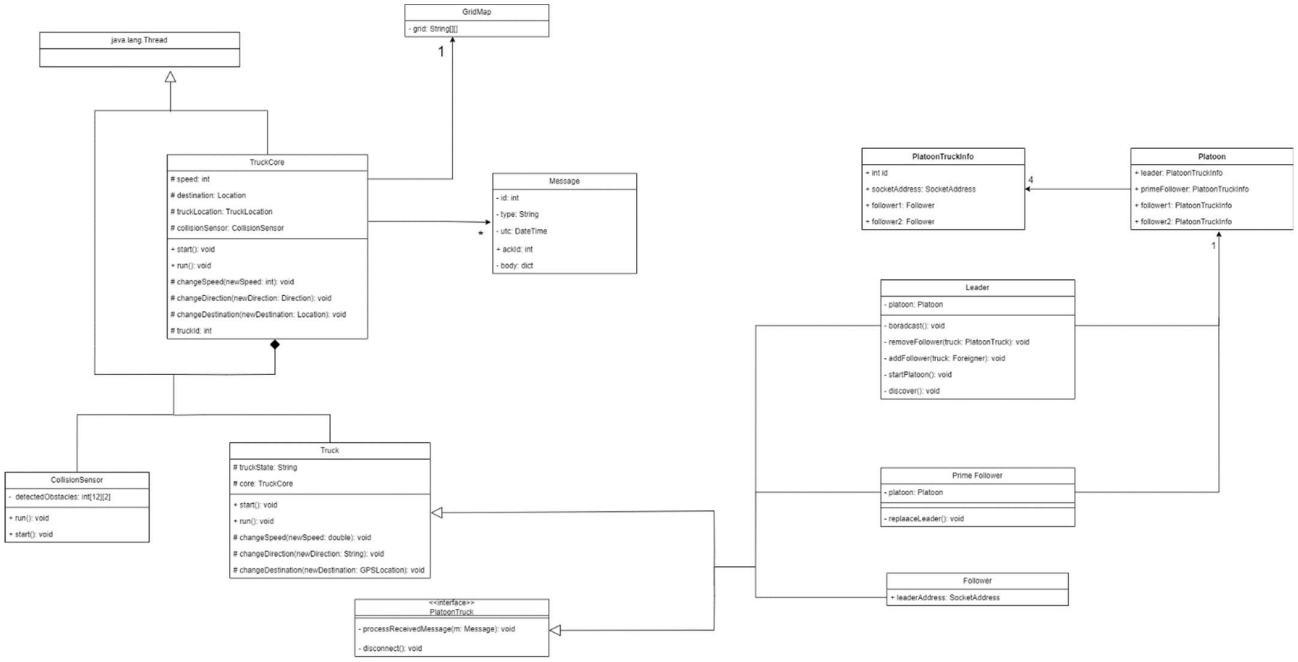


Fig. 14. HMI Module internal structure

III. Design and Implementation

After the platoon model and its relevant concepts were defined and decided upon, it was time to concretize some of its aspects and put some constraints on the model and surrounding concepts. This would allow for the design of a definitive model that could be implemented for a simulation. As seen in Figure 15, the system model sees some major changes along with keeping the base concepts of the platoon model.

Fig. 15. Design¬-level class diagram



A. Programming Language

The Java language was chosen as the programming language for the implementation. Java is a high-level, object-oriented programming language that allows for applying the concepts of *object-oriented programming* to use. Furthermore, thanks to its Java Virtual Machine (JVM), its code can be run on any machine that can run its virtual machine. This allows for easy simulation and speedy implementation of the concepts; at the same time, it has a similar syntax style to that of C and C++, and its code can then easily be translated to those languages for concrete implementations on specific hardware [6]. Lastly, for the team members, choosing this programming language for this project served as a suitable choice to gain a foothold in the Java language and put the teachings of relevant courses to use.

B. Simulation Space

Normally, the simulation space for a truck platooning project would be a subset of the real world. This is also mirrored in the UML class diagram in the analysis phase; In that diagram, we can observe that for attributes such as *Truck* location, a *GPSLocation* type was used. Though this specification is per the real world, its simulation especially a visual one would be out of the scope for this project. So instead, a two-dimensional grid map was agreed upon that would allow for easy simulation, visualization, and also testing of the concepts. The downside for this choice is that such a simulation space would mean discrete values for concepts such as speed and location; but as the goal of this project is to implement a distributed, parallel system, this matter was deemed negligible.

In our grid map, each truck shall take three consecutive, vertical cells, one for a truck head and two for its tail. The optimal distance between a truck's tail and the following truck's head shall be three empty vertical cells. Trucks may go in one of three directions: north, northwest, and northeast. As speed, trucks may go up to four vertical cells per second. In case of a direction other than north, the trucks may only

change one column per second, but they will progress vertically per their speed. For a journey through the map, the trucks start from the lower rows of the map and progress through the cells until they reach their desired row and column. Using a collision sensor, a truck may only see one column to its sides and up to two rows in front of it. For example, a truck's head located at row two and column two would be able to detect objects in the following cells: in column one, rows zero to four would be observable; in column two, rows zero to one would be observable; in column three (like column one), rows zero to four would be observable.

C. Communication

The platooning model uses the TCP/IP as the communication protocol. Through its *java.net* package, Java already provides interfaces for socket programming using TCP/IP (denoted by the *Socket* and the *ServerSocket* classes) [7]. Combined with Java's inherent support for threads, the distributed architecture of TCP/IP over threads is an appropriate choice for the simulation. Each truck would be represented by a TCP/IP socket and it would occupy a unique port on the same computer (each run via a separate thread). This would not only allow the simulation of a distributed architecture but also let the simulation be run on the same computer. Although this choice would ignore real-world, network-related issues such as latency, it provides an easily configurable platform to implement and test our distributed system.

Similar to a distributed system, the trucks inside the map have little to no information about the other trucks. The only assumption is that the Leader at the beginning of the simulation would receive the addresses for the other trucks in order to initiate communication with them. Other than that, all the other trucks and the Leader itself have no information about the other trucks. The only way for the trucks to exchange data with each other, as mentioned before, is the passing of messages between each other.

D. Role Switching

As soon as the decision for a programming language was made, it was clear that a change of roles based on the defined models would not be a straightforward transition. Each role within the platoon model has its own class within the class diagram and eventually, trucks would need to change roles. If each role is denoted by a certain class, then how would a truck change its role and consequently its object class type? This brought us to two possible solutions: roles as a state machine vs. roles as threads.

1) Roles as State Machine

For this solution, there would have to be a complete Truck object that handles all role types. This solution would result in a large Truck object that would contain methods and logic for all role types (Leader, Follower, PrimeFollower, and a normal Truck itself). This solution would be closer to an embedded program as the program would only keep states and change between them when necessary [8]. However, the downside is that the resulting implementation would not be as structured as one would achieve with an OOP approach.

2) Roles as Threads

In the second solution, each role would be an object which could be run as a separate thread. Then, as a truck changes a role, it would destroy its existing role thread and create another one. For example, when a truck has the role of a Follower, it has a long-running thread of type Follower; when the truck needs to become a PrimeFollower, it destroys the Follower thread and creates a PrimeFollower thread instead. This solution results in a much more structured code where OOP concepts such as inheritance, polymorphism, etc. are followed. The downside of the solution is that the creation and teardown of threads bring their own overhead.

Finally, the second solution (roles as threads) was chosen as the implementation method. As Java was the programming language and the goal of the project was simulating a distributed, parallel system and not a concrete embedded system, the second solution was a more suitable choice considering the goals of the project.

E. Node Implementation

For the implementation, each truck would be a long-running thread on its own that would communicate with other trucks (threads) using their socket addresses. As discussed in the Communication sub-chapter, trucks would not need any information from each other at the start of the simulation (except for the Leader). So far, there would be no shared resources between the trucks. The exception is that of the grid map. The grid map was implemented as its own class called *GridMap*. For trucks to access information regarding their location, they would need to access the GridMap object. This object is the only shared resource between the different truck threads.

Per the decision to have roles as threads, some changes became necessary to the overall structure of the model. Evidently, a thread object cannot create itself; a thread has to be created by other threads. While a role thread could create another thread for another role, this approach is not only unclean, it could also result in undefined behavior. The better approach is to have another thread handle this transition. Therefore a *TruckCore* class was created that would handle the communication and role transition. The core keeps essential information regarding received messages, speed and location information, its state, and its role.

Let us go through the transition from a Truck to a PrimeFollower. Except for the Leader, each core assumes the role of Truck at the beginning of the simulation; it waits for a discovery message for a certain time, otherwise, it quits. When a discovery message is received, the core adds this message to the list of messages. The role processes the messages on each iteration and decides accordingly. After the Truck role finalizes joining the platoon and has its assigned role at hand, it sets the final message for reference in *Truck.referenceMessage*, sets the relevant core state in *TruckCore.state* and exits. Consequently, in the next iteration of the TruckCore, besides handling the communication, the core checks to see whether the role thread is alive or not (see). If it is not alive, the core investigates the reason. After checking the set state and reference message, the core realizes that the Truck role has finished and the truck needs to become a Follower now. The core starts creating a new Follower class, sets it to its *TruckCore.role* property, and starts its thread. This transition process (see Fig. 16) applies to all possible role changes.

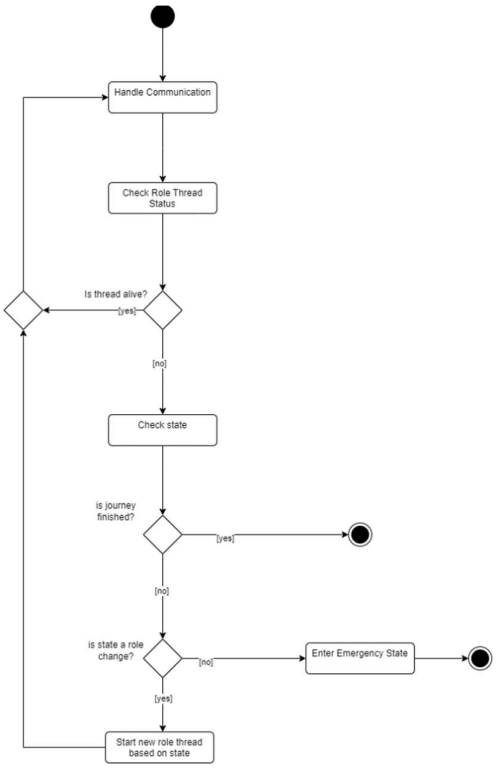


Fig. 16. Activity diagram of how a TruckCore manages its role

IV. CONCLUSION

In summary, our project on truck platooning represents a significant step forward thanks to technological progress. Platooning involves coordinating multiple trucks to travel closely together, offering several benefits such as saving fuel, reducing carbon emissions, and improving fleet management efficiency.

During the planning and design phase of our project, a model-driven approach was taken to realize the platoon model's concept and related behavior. The diagrams produced during the modeling stages helped us map out the interactions and features of our four-truck platoon system: the Leader, Prime Follower, and Follower trucks. We meticulously analyzed the requirements, defined block structures, and illustrated internal block relationships to ensure a comprehensive understanding of system dynamics.

From a technical perspective, our system relies on the TCP/IP for communication between trucks. We implemented the system using Java, a versatile programming language known for its reliability and scalability. Utilizing the robustness of TCP/IP and employing socket addresses for communication, we established seamless communication channels to facilitate real-time data exchange among trucks, enhancing operational efficiency.

In terms of spatial orientation, we opted for a simpler approach by using a two-dimensional grid map instead of complex GPS coordinates. This decision not only simplifies system implementation but also improves usability and clarity for participants. By representing truck positions within a grid framework, we provide an intuitive visualization tool that aids in simulation and decision-making processes.

Overall, our project highlights not only technological advancements but also a commitment to sustainability and efficiency in transportation. By utilizing technology to promote collaborative and environmentally friendly practices, we aim to contribute to a future where logistical operations are not only optimized but also environmentally responsible.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to Dr. Prof. Stefen Henkler for his constant guidance and support which helped us to understand the materials presented within the Distributed and Parallel Systems course and put them to use appropriately.

REFERENCES

[1] J. Muvuna, T. Boutaleb, K. J. Baker and S. B. Mickovski, "A Methodology to Model Integrated Smart City System from the Information Perspective," *Smart Cities,* vol. 2, no. 4, pp. 496-511, November 2019.

[2] A. Tanenbaum and M. Van Steen, Distributed systems, Upper Saddle River: Prentice-hall, 2017.

[3] N. J. Alpern and R. J. Shimonski, "CHAPTER 1 - Network Fundamentals," in *Eleventh Hour Network+*, 2010, pp. 1-18.

[4] M. Strunz, J. Heinovski and F. Dressler, "CoOP: V2V-based Cooperative Overtaking for Platoons on Freeways," in *IEEE International Intelligent Transportation Systems Conference (ITSC)*, Indianapolis, 2021.

[5] M. Mahani, D. Rizzo, C. Paredis and Y. Wang, "Automatic Formal Verification of SysML State Machine Diagrams for Vehicular Control Systems," *SAE Int. J. Adv. & Curr. Prac. in Mobility,* vol. 3, no. 5, p. 2272–2280, April 2021.

[6] D. J. E. H. a. W. S. Colleges, Introduction to Programming Using Java, 2021.

[7] Oracle, "java.net (Java Platform SE 8)," 08 01 2024. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/net/package-summary.html. [Accessed 28 01 2024].

[8] M. Gomez, "Embedded State Machine Implementation," *Embedded Systems Programming,* vol. 13, no. 13, p. 40, December 2000.

AFFIDAVIT

We (Mohammed Rizwan, Hazhir Amiri, Anguiga Hermann) herewith declare that we have composed the present paper and work ourselves without the use of any other than the cited sources and aids. Sentences or parts of sentences quoted are marked as such other references about the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form have not been submitted to any examination body and have not been published. This paper was not yet, even in art used in another examination or as a course performance.

Date: 30.01.2024

Name	Signature
Mohammed Rizwan	
Hazhir Amiri	
Anguiga Hermann	

ANNEX

A. Source Code

The following table contains a list of all the classes defined within the project with their relevant information.

Package Name	Class Name	Extends/Implements	Description
dps	CollisionSensor	Java.land.Thread	Checks the surrounding cells of its TruckCore owner and reports obstacles within these cells (partially implemented).
	Main	-	The main file is run when the program starts. Initializes the simulation.
	Message	-	Represents a message that is sent between trucks.
	Utils	-	Contains utils method used throughout the project by different classes.
dps.platoon	Follower	Truck	Contains the logic for a Follower role.
	Leader	Truck	Contains the logic for a Leader role.
	Platoon	-	Contains information regarding the platoon members in the form of PlatoonTruckInfo objects.
	PlatoonTruckInfo	-	A data type that contains the ID and the socket address of a truck.
	PlatoonTruck	-	Unused interface from the design stage.
	PrimeFollower	Truck	Containts the logic for the Prime Follower role.
dps.truck	DatedTruckLocation	TruckLocation	A data type contains the TruckLocation of a truck along with the time and date the object wa created. Follower trucks use this information to calculate the location of the Leader.
	SocketAddress	-	A data type that contains the IP address and port of a truck.
	Truck	java.lang.Thread	Contains the logic for a truck that is part of no platoon. Also defines overall truck methods any other kind of truck can also use. It is synanymous to Foreigner and Truck classes in the analysis class diagram.
	TruckClient	-	A static class that is used to send socket messages.
	TruckCore	java.lang.Thread	Handles truck communication and role transitions. It is the only part of a truck that stays alive throughout the simulation. Contains information that is retained throughout role transitions, like speed, direction, etc.
	TruckLocation	-	A data type that contains the location and directon of a truck.
dps.map	Direction	-	An *enum* type that contains possible truck directions: NORTH
	GridMap	java.awt.JFrame	Contains the two-dismensional array of the map elements and is responsible for updating and visualizing the map.
	Location	-	A data type that holds the row and column.
	TruckInfo	-	A data type that holds necessary information needed by the GridMap.

B. GitHub Overview

1) Lines of code: The implementation has a total of 2275 lines of code that spans 19 files.

2) Number of submits per person: Considering all commits except merge commits, across all branches, the project members had the following numbers: Hazhir Amiri, with the username *allerter*, had 156 commits. Mohammad Rizwan, with the usernames *Mohammed Rizwan* and *MR-KHAN0099* had 39 commits. Anguiga Hermann, with the username *H.Anguiga2023*, had 30 commits.

3) Structure: The project has the following hierarchy:

a) Root: The topmost folder contains the PDF version of this article, the folders below, and a *README.md* file with instructions for running the simulation.

b) Diagrams: This folder contains pictures and the editable versions of the diagrams used throughout this article and the modeling stages.

c) Implementation: This folder contains the simulation implementation.

d) Docs: This folder contains the editable version of the documents, including this article.

C. Contribution

Member Full Name	**Contribution in Percentages**	**Work Estimation in Hours**	**Chapters Written in the Paper**
Hazhir Amiri	35%	25	**Design and Implementation, Annex**
Mohammed Rizwan	35%	25	**Abstract**, Error! Reference source not found. (except Requirements), **Platoon Model** (except System Architecture and some parts in UML Digrams), Figures 2, 3, 7, 8, and 9 in **UML Digrams** and their description.
Anguiga Hermann	30%	35	**Requirements**, Figures 4, 5, 6, 10, 11, 12, 13, and 14 in **UML Digrams** and their descriptions, **System Architecture**, Error! Reference source not found.