Truck Platooning Report

Shihab Ud Doula Email: shihab-ud.doula@stud.hshl.de
Lippstadt, Germany

Jasmeet Singh Matta Email: jasmeet-singh.matta@stud.hshl.de
Paderborn, Germany

Moshiur Rahman Prince Email: moshiour-rahman.prince@stud.hshl.de
Lippstadt, Germany

Contents

Abstract: This paper presents the development and implementation of a truck platooning system aimed at enhancing track-based transportation safety, efficiency, and reliability. It addresses various scenarios, including coupling-decoupling, lane changing, steering, and obstacle handling within single driver and system models. Methodologies employed include UML diagrams for initial design, UPPAAL models for formal verification, and Python implementations for practical simulation and validation. The results demonstrate that the system operates without deadlocks and achieves high accuracy in leader vehicle determination, thereby reducing the risk of accidents and improving traffic flow.

1 Motivation

The track platooning project focuses on developing and implementing automated systems for managing and optimizing the behaviour of platoons of vehicles on tracks. This document outlines the project's objectives, methodologies, and outcomes, detailing the detection and handling of various scenarios such as coupling-decoupling, lane changing and steering, single driver and system model with obstacles. This project aims to enhance track-based transportation systems' safety, efficiency, and reliability. By automating the detection and management of critical scenarios, we aim to reduce the risk of accidents, improve traffic flow, and minimize the need for manual intervention. This project leverages advanced modelling techniques and real-time data processing to achieve these goals. In this project, we will design UML diagrams, develop UPPAAL models for formal verification, and implement Python-based simulations to validate the system's performance.[Zh24, TJS16]

2 Literature Review

Track platooning has been a research subject in road and rail transportation. Previous studies have explored various aspects of platooning, including communication protocols, control algorithms, and safety mechanisms. The literature reveals the potential of using UML and UPPAAL models to formalize system behaviour and ensure robust performance. This project builds on these foundations by integrating scenario-specific models and implementing them in Python for real-world applicability.[Ku11]

3 Problem Statement

The project addresses several critical scenarios vital for the successful implementation of track platooning:

- **Obstacle Avoidance**: To detect obstacles and apply brakes accordingly to slow the vehicle or stop it down ultimately.
- **Lane Changing and Steering**: Managing lane changes and steering actions to prevent collisions and ensure smooth transitions.
- **Environmental Model**: Simulating the behaviour of an overall system and how the system will react to the changes in the environment

These Implementations are shown in the following sections.

4 Modeling Scenarios with UML

4.1 Requirements Analysis for Truck Platooning

The requirements diagram in Figure 1 outlines the components of the Autonomous Truck Platooning System. It includes leading and following vehicles, the central hub, and the environment. Each element has specific requirements, such as platoon formation, communication, management, and handling emergency scenarios like connection loss, GPS failure, malfunctions, and cyberattacks.

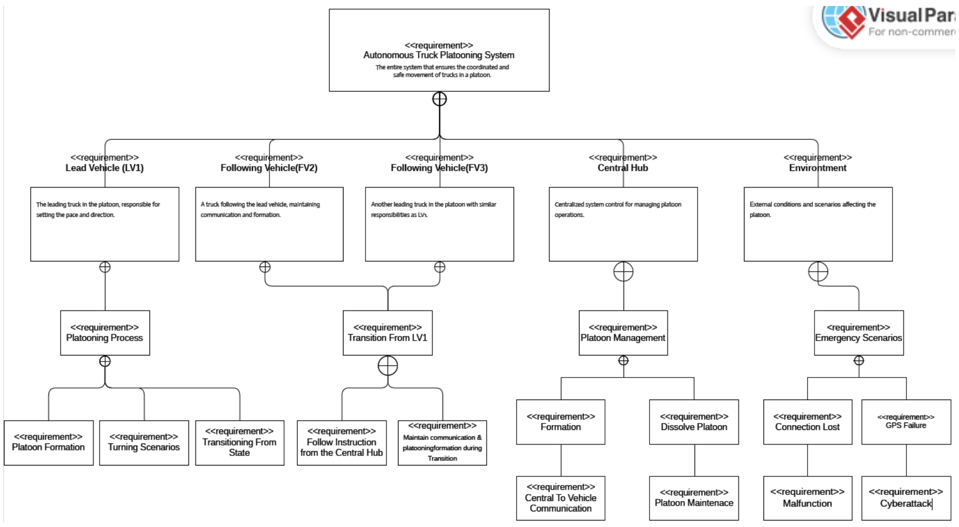


Fig. 1: Requirements Diagram For The System

4.2 Scenario 1(Obstacle Avoidance)

Figure 2 is a state machine diagram that shows how the system acknowledges obstacles and behaves accordingly.

4.3 Scenario 2(Lane Changing And Steering)

This sequence diagram in Figure 3 shows a driver requesting lane changes and the system's response. It involves checking lane availability, executing the lane change, and returning to the original lane, highlighting the system's ability to manage steering and lane changes dynamically.

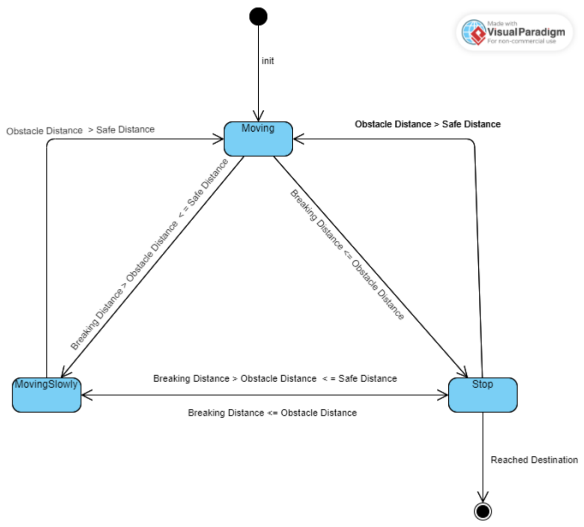

init
Moving
Obstacle Distance > Safe Distance
Obstacle Distance > Safe Distance
Breaking Distance > Obstacle Distance <= Safe Distance
Breaking Distance <= Obstacle Distance
MovingSlowly
Stop
Breaking Distance > Obstacle Distance <= Safe Distance
Breaking Distance <= Obstacle Distance
Reached Destination

Fig. 2: State Machine Diagram for Obstacle Avoidance

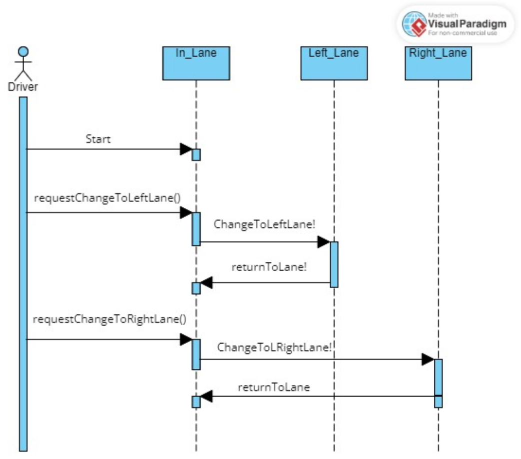
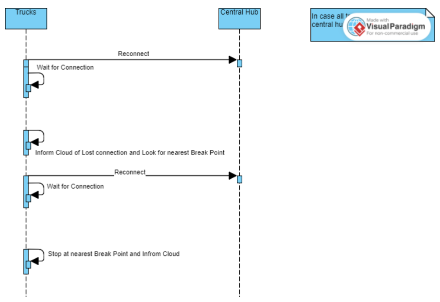


Fig. 3: Lane changing and steering Sequence diagram

4.4 Scenario 3(Environmental Interactions)

The diagrams in Figure 4 & 5 depict how trucks manage lost connections and emergencies. They show reconnecting, informing the cloud, and handling emergencies by sending instructions and coordinating responses, ensuring the system's resilience and safety in dynamic environments.


Trucks
Central Hub
In case all t
central hu
Reconnect
Wait for Connection
Inform Cloud of Lost connection and Look for nearest Break Point
Reconnect
Wait for Connection
Stop at nearest Break Point and Infrom Cloud
Fig. 4: Lost connection Sequence diagram

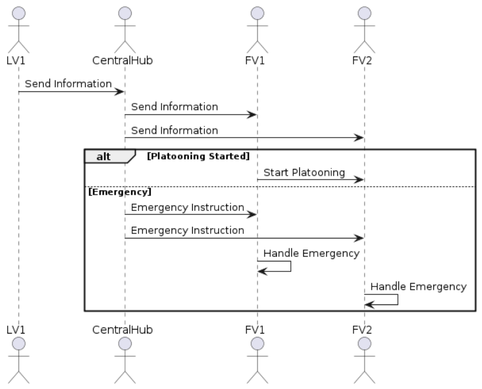


Fig. 5: Emergency Sequence diagram

Additional UML diagrams are available on GitHub to better understand the system. Visit **Scenarios with UML Diagram**

5 Formal Verification with UPPAAL

We have completed the UPPAAL model for three scenarios, as mentioned, ensuring that the system operates without deadlock and can dynamically respond to environmental changes while maintaining safe operations.

5.1 UPPAAL Model: Obstacle Avoidance

The models in (Figures 6 and 7) depict a truck detecting and responding to obstacles. The truck moves, detects obstacles, adjusts its speed, and decides to stop or slow down based on the obstacle's distance. This ensures safe navigation through dynamic environments. The leader truck and the following trucks coordinate to manage obstacles, prevent collisions and manage traffic flow. For the complete XML file, visit our GitHub repository for **Obstacle Avoidance Model.**

5.2 UPPAAL Model: Lane Changing And Steering

The model in Figure 8 illustrates how a vehicle changes lanes. It includes checking lane availability, executing lane changes, and ensuring safe transitions back to the original lane. The model ensures that lane changes are conducted smoothly without disrupting the overall flow of the platoon. For the complete XML file, visit our GitHub repository for **Lane Change and Steering Model**.

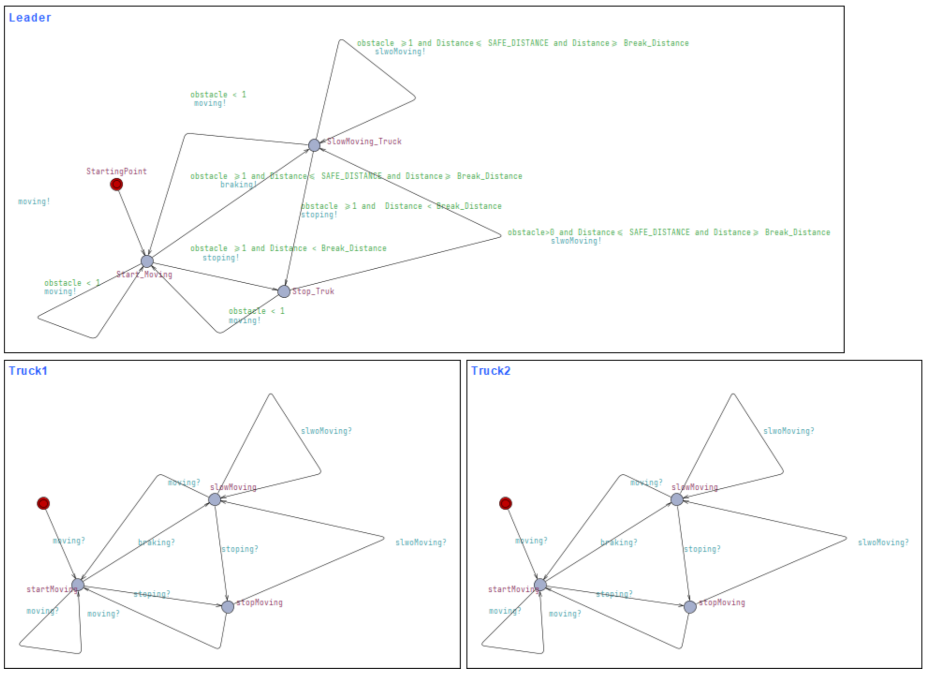


Fig. 6: Obstacle Uppaal 1

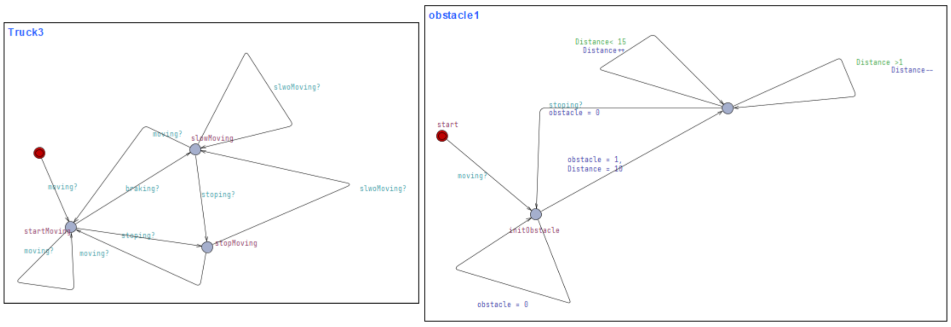


Fig. 7: Obstacle Uppaal 2

5.3 UPPAAL Model: Environmental Conditions

The model in Figure 9 shows how components interact within various environmental conditions, including lane changing, obstacle avoidance, and platoon formation. The model ensures that vehicles can dynamically respond to environmental changes and maintain safe operations.The controller regulates the number of trucks in a platoon. Two conditions are checked during the joining of a truck in platoon formation phase. Follower trucks can join if:

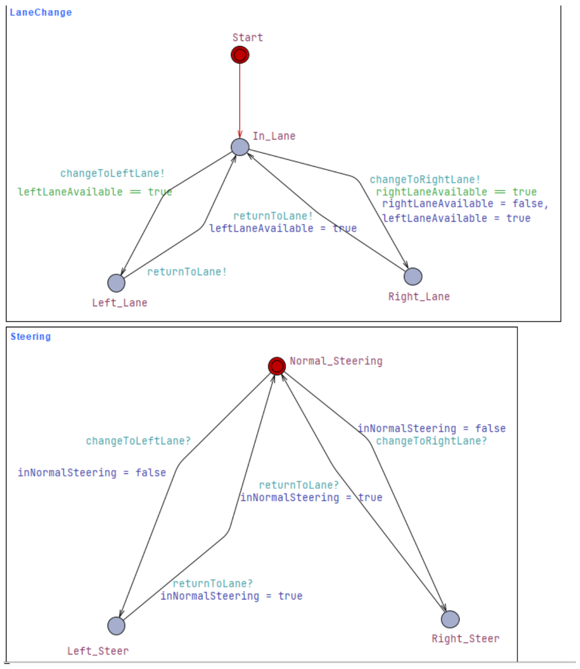


Fig. 8: Lane Change and Steering Uppaal

1. Number of trucks in the platoon is greater than 0 (this is to ensure that a platoon formation is not possible with negative nTrucks values)

2. Number of trucks in the platoon is smaller than MAX_Trucks which is declared as an integer in the global declarations. The value determines the maximum number of trucks including the leader truck that can join the platoon.

For the complete XML file, visit our GitHub repository for **Environmental Model.**

5.4 Complete UPPAAL Model: Final Simulation

The complete UPPAAL model integrates all scenarios, ensuring the platoon's coordinated behaviour under various conditions. The main model combines all individual components, such as the Leader Truck, Follower Truck, Controller, and Roadside Infrastructure, into a unified system. This integrated approach validates the system's performance, emphasizing the dynamic interaction between components under varying environmental conditions.

Main Model Overview: Figure 10 represents the combined main model, which integrates the different UPPAAL scenarios to simulate the complete truck platooning system. The

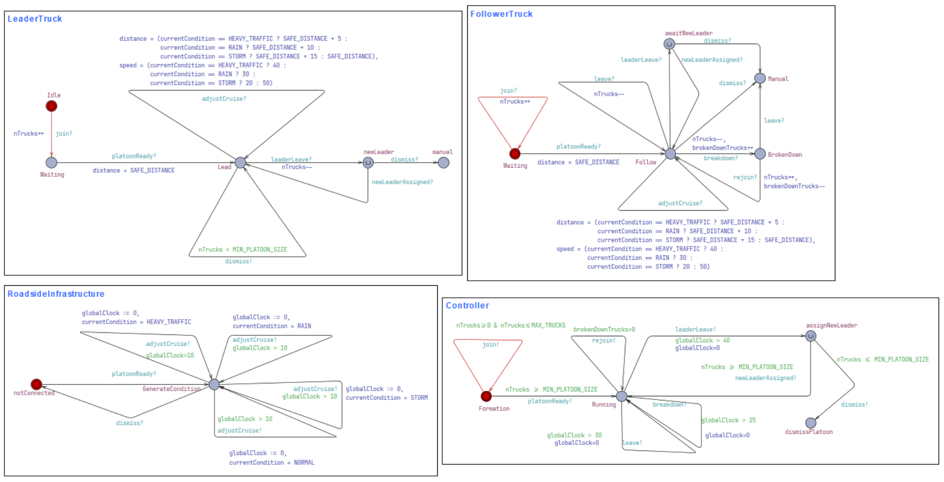


Fig. 9: Environmental Model

main model captures the interactions and transitions necessary for coordinated platoon behaviour.

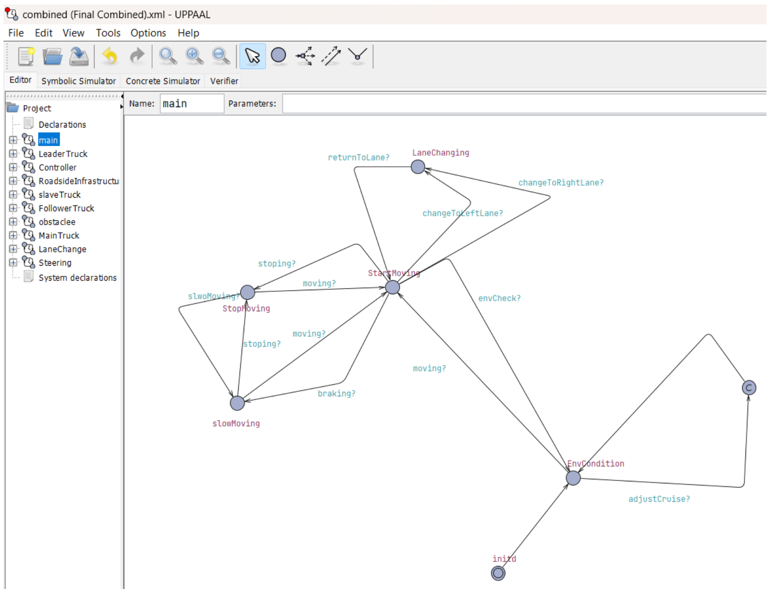
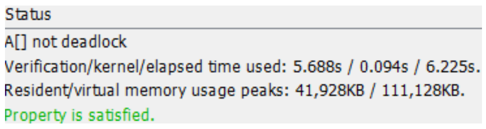


Fig. 10: Main Model of the complete system

Verification Status: The final verification status confirms that the UPPAAL model is free from deadlocks, as shown in Figure 11, ensuring reliable system performance. The model successfully satisfies all specified properties, validating the truck platooning system's robustness and safety.



```
Status
A[] not deadlock
Verification/kernel/elapsed time used: 5.688s / 0.094s / 6.225s.
Resident/virtual memory usage peaks: 41,928KB / 111,128KB.
Property is satisfied.
```

Fig. 11: UPPAAL model verified of having no deadlocks

The UPPAAL models for obstacle avoidance, lane changing, and environmental conditions have been successfully implemented. The final integrated model demonstrates the system's ability to handle complex interactions and maintain safety and efficiency in various scenarios, validating the robustness of our truck platooning system under diverse conditions. This comprehensive approach ensures that the system meets the desired performance and safety standards, providing a reliable solution for automated truck platooning.[BDL04, BY03]

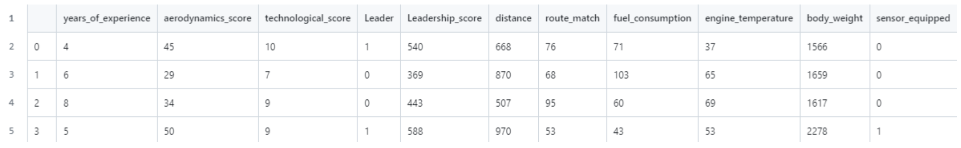
For the complete simulation files, including all UPPAAL XML files, visit our GitHub repository. **Combined Model**

6 Machine Learning for Leader Vehicle Determination

This section explores various machine learning algorithms to determine the leader vehicle in our truck platooning system. Using a consistent dataset to compare their effectiveness, we implement Decision Tree, Logistic Regression, and linear regression models.

6.1 Dataset Overview

The dataset in Table 1 used for the machine learning models includes the following features:



1		years_of_experience	aerodynamics_score	technological_score	Leader	Leadership_score	distance	route_match	fuel_consumption	engine_temperature	body_weight	sensor_equipped
2	0	4	45	10	1	540	668	76	71	37	1566	0
3	1	6	29	7	0	369	870	68	103	65	1659	0
4	2	8	34	9	0	443	507	95	60	69	1617	0
5	3	5	50	9	1	588	970	53	43	53	2278	1

Fig. 12: Sample Dataset

- Years_of_experience: For a single truck, the total number of years in operation or service.
- aerodynamics_score: Aerodynamics score is a numerical value indicating how well a truck's design limits air resistance.
- techonoligical _score:A technological score is a numerical evaluating that rates the amount and efficacy of modern technology and features built into a truck.

- Route Distance: The total distance of the route in kilometres.
- Route Match: The percentage matches the desired route.
- Consumption: Fuel consumption in litres.
- Body Characteristics: Numeric representation of vehicle body traits.
- Equipment Sensors: Numeric representation of the quality and type of sensors.
- Leader Vehicle: Binary value indicating if the vehicle is a leader (1) or not (0).

Sample Data: Visit our project repository on GitHub at **Dataset**.

6.2 Model Implementations

We implemented three machine learning models to determine the leader vehicle: Decision Tree, Logistic Regression, and linear regression.

6.2.1 Decision Tree Classifier

A decision tree classifier was implemented using Scikit-learn in Google Colab.

Implementation Summary:

- Model Training: A `DecisionTreeClassifier` was initialized and trained on the training set.
- Accuracy: The decision tree achieved an accuracy of 93.5%.
- Implementation available in Github: **Leader selection by Decision Tree**.

6.2.2 Logistic Regression

Logistic regression was also implemented to compare its performance with the decision tree classifier.

Implementation Summary:

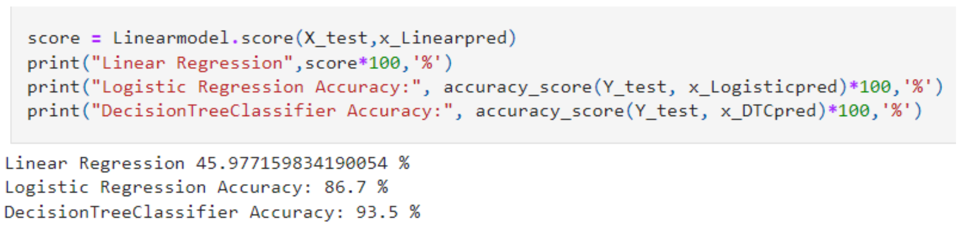
- Model Training: A `LogisticRegression` model was trained on the training set.
- Accuracy: The logistic regression model achieved an accuracy of 86.7%.
- Implementation available in Github: **Leader Selection by Logistic Regression**

6.2.3 Comparative Analysis of Machine Learning Algorithms

Although typically used for prediction, linear regression was tested as a baseline for classification.

Implementation Summary:

- Model Training: A `LinearRegression` model was trained on the training set.
- Accuracy: The linear regression model achieved an accuracy of 45.98%.
- Implementation available in Github:**Comparison Between ML Algorithms**



```
score = Linearmodel.score(X_test,x_Linearpred)
print("Linear Regression",score*100,'%')
print("Logistic Regression Accuracy:", accuracy_score(Y_test, x_Logisticpred)*100,'%')
print("DecisionTreeClassifier Accuracy:", accuracy_score(Y_test, x_DTCpred)*100,'%')
```

```
Linear Regression 45.977159834190054 %
Logistic Regression Accuracy: 86.7 %
DecisionTreeClassifier Accuracy: 93.5 %
```

Fig. 13: Different accuracy scores of different ML models tested on the same set of Data

Accuracy Comparison:

- Decision Tree Classifier Accuracy: 93.5%
- Logistic Regression Accuracy: 86.7%
- Linear Regression Accuracy: 45.98%

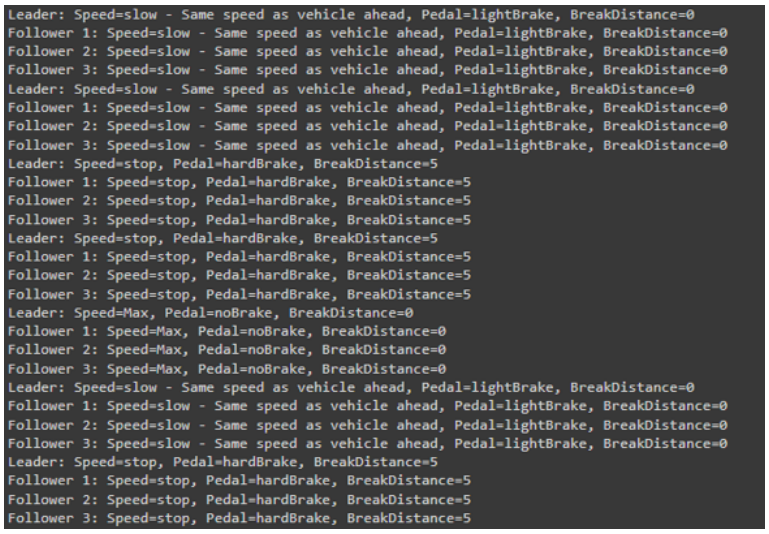
The decision tree classifier achieved the highest accuracy, making it the best-suited algorithm for classifying the leader vehicle in our platooning system observed from Fig. 13. Logistic regression is a robust alternative, while linear regression is unsuitable for this classification task.[SZL16, IAH20]

7 Developing the Simulation Environment

To validate the effectiveness of our truck platooning system, we developed a comprehensive simulation environment using Python. This environment integrates multiple scenarios, allowing us to observe and analyze the system's behaviour under various conditions. Initially, we implemented each scenario in a separate Python model which is given as follows:

7.1 Obstacle Avoidance

- Logic: The trucks adjust their speed or stop based on the detected proximity and type of obstacle.
- Key Results: The system successfully avoided obstacles, as shown in the simulation output.
- Implementation available in Github: **Obstacle Avoidance**



```
Leader: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 1: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 2: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 3: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Leader: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 1: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 2: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 3: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Leader: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 1: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 2: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 3: Speed=stop, Pedal=hardBrake, BreakDistance=5
Leader: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 1: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 2: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 3: Speed=stop, Pedal=hardBrake, BreakDistance=5
Leader: Speed=Max, Pedal=noBrake, BreakDistance=0
Follower 1: Speed=Max, Pedal=noBrake, BreakDistance=0
Follower 2: Speed=Max, Pedal=noBrake, BreakDistance=0
Follower 3: Speed=Max, Pedal=noBrake, BreakDistance=0
Leader: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 1: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 2: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 3: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Leader: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 1: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 2: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 3: Speed=stop, Pedal=hardBrake, BreakDistance=5
```

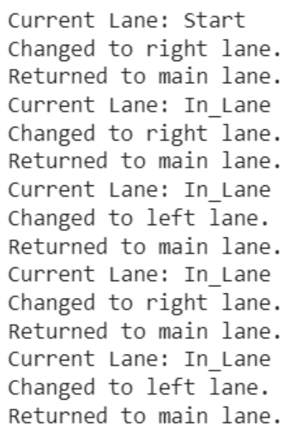
Fig. 14: Obstacle Avoidance Simulation

7.2 Lane Changing and Steering

- Logic: Trucks change lanes based on lane availability and traffic conditions, returning to the main lane when appropriate.
- Key Results: The trucks performed lane changes smoothly, as indicated in the output log.
- Implementation available in Github: **Lane Changing and Steering**

7.3 Environmental Model

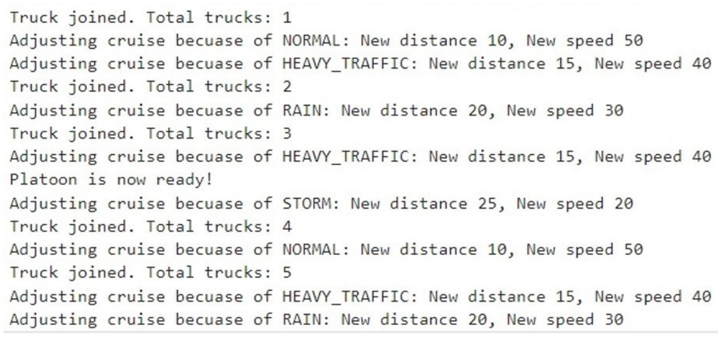
- Logic: The system adjusts speed and following distance based on weather conditions like heavy traffic, rain, and storms.



```
Current Lane: Start
Changed to right lane.
Returned to main lane.
Current Lane: In_Lane
Changed to right lane.
Returned to main lane.
Current Lane: In_Lane
Changed to left lane.
Returned to main lane.
Current Lane: In_Lane
Changed to right lane.
Returned to main lane.
Current Lane: In_Lane
Changed to left lane.
Returned to main lane.
```

Fig. 15: Lane Changing And Steering Simulation

- Key Results: Trucks adjusted their behaviour accurately to match environmental changes.
- Implementation available in Github: **Environmental Model**



```
Truck joined. Total trucks: 1
Adjusting cruise becuase of NORMAL: New distance 10, New speed 50
Adjusting cruise becuase of HEAVY_TRAFFIC: New distance 15, New speed 40
Truck joined. Total trucks: 2
Adjusting cruise becuase of RAIN: New distance 20, New speed 30
Truck joined. Total trucks: 3
Adjusting cruise becuase of HEAVY_TRAFFIC: New distance 15, New speed 40
Platoon is now ready!
Adjusting cruise becuase of STORM: New distance 25, New speed 20
Truck joined. Total trucks: 4
Adjusting cruise becuase of NORMAL: New distance 10, New speed 50
Truck joined. Total trucks: 5
Adjusting cruise becuase of HEAVY_TRAFFIC: New distance 15, New speed 40
Adjusting cruise becuase of RAIN: New distance 20, New speed 30
```

Fig. 16: Environmental Model Simulation

7.4 Merging Scenarios into a Complete Simulation

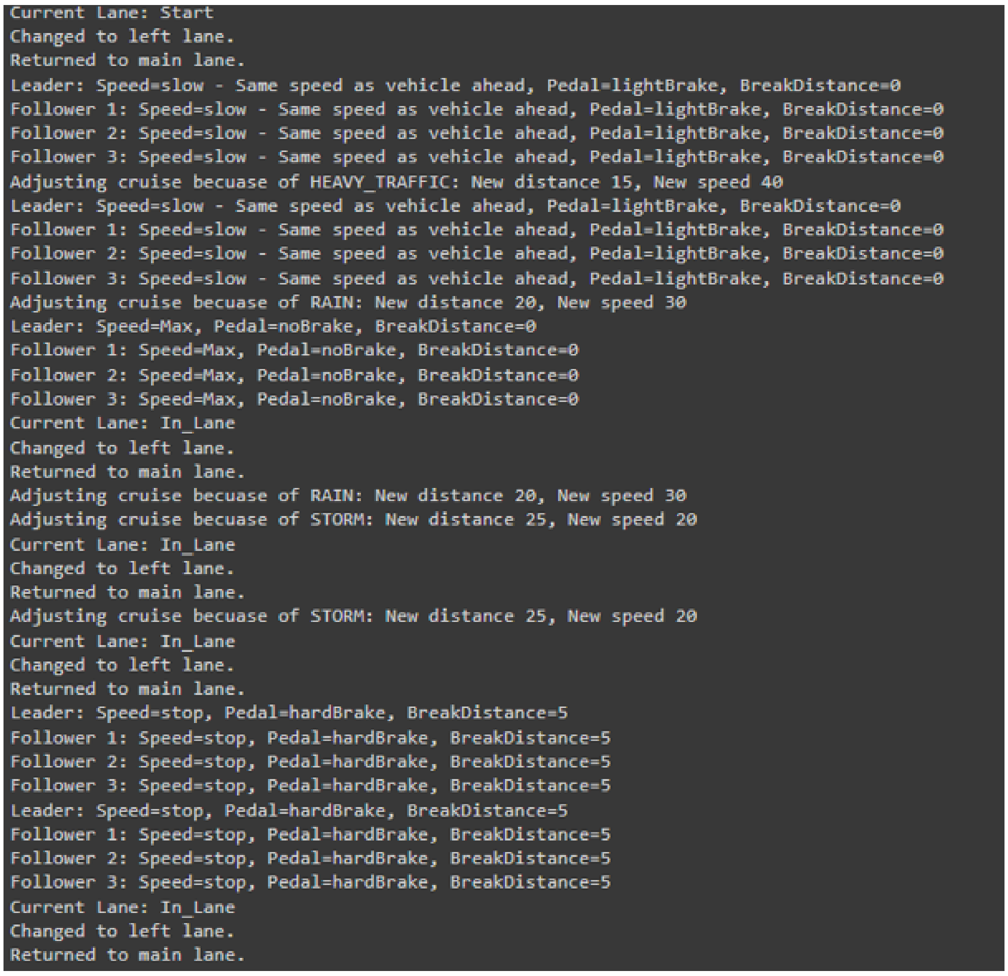
After developing and validating individual scenarios, we integrated them into a unified simulation environment.

Integration Summary:

Approach: Combined the logic from each scenario into a single Python script.

- Challenges: Addressed issues like synchronising different behaviours and ensuring smooth transitions between scenarios.
- Solutions: Used modular coding practices and thorough testing to resolve integration issues.
- Complete Simulation is available on Github: **Final Simulation in Python**

Complete Simulation Output:



```
Current Lane: Start
Changed to left lane.
Returned to main lane.
Leader: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 1: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 2: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 3: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Adjusting cruise becuase of HEAVY_TRAFFIC: New distance 15, New speed 40
Leader: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 1: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 2: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Follower 3: Speed=slow - Same speed as vehicle ahead, Pedal=lightBrake, BreakDistance=0
Adjusting cruise becuase of RAIN: New distance 20, New speed 30
Leader: Speed=Max, Pedal=noBrake, BreakDistance=0
Follower 1: Speed=Max, Pedal=noBrake, BreakDistance=0
Follower 2: Speed=Max, Pedal=noBrake, BreakDistance=0
Follower 3: Speed=Max, Pedal=noBrake, BreakDistance=0
Current Lane: In_Lane
Changed to left lane.
Returned to main lane.
Adjusting cruise becuase of RAIN: New distance 20, New speed 30
Adjusting cruise becuase of STORM: New distance 25, New speed 20
Current Lane: In_Lane
Changed to left lane.
Returned to main lane.
Adjusting cruise becuase of STORM: New distance 25, New speed 20
Current Lane: In_Lane
Changed to left lane.
Returned to main lane.
Leader: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 1: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 2: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 3: Speed=stop, Pedal=hardBrake, BreakDistance=5
Leader: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 1: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 2: Speed=stop, Pedal=hardBrake, BreakDistance=5
Follower 3: Speed=stop, Pedal=hardBrake, BreakDistance=5
Current Lane: In_Lane
Changed to left lane.
Returned to main lane.
```

Fig. 17: Combines Scenarios Simulation in Real Time

The complete simulation provided valuable insights into the performance and robustness of our truck platooning system.[Pe11]

Key Observations:

- Obstacle Avoidance: Trucks effectively detected and responded to obstacles, maintaining safety.
- Lane Changing: The system handled lane changes efficiently, preventing potential collisions.
- Environmental Adaptation: Trucks adjusted their behaviour dynamically based on changing environmental conditions, demonstrating system adaptability.

8 Conclusion

In this project, we developed an advanced truck platooning system, focusing on crucial scenarios like obstacle avoidance, lane changing, and environmental adaptation. Using UML and UPPAAL, we formalized system behaviours and ensured robust performance through formal verification. Our machine learning models, particularly the decision tree classifier, effectively identified leader vehicles, enhancing platoon efficiency. Finally, we implemented a comprehensive Python-based simulation environment to integrate and test all scenarios. The simulation validated our system's safety, adaptability, and efficiency, demonstrating the potential of automated truck platooning in real-world applications. This project showcases our ability to design, model, and validate complex autonomous systems.

9 Contribution to the Project

- GitHub Repository for: **Truck Platooning**
- **Jasmeet Singh Matta** - Obstacle Avoidance (UML Diagram, UPPAAL model, Python code)
  Leader Selection: Linear Regression
- **Shihab Ud Doula** - Lane Change and Steering (UML Diagram, UPPAAL model, Python code)
  Leader Selection: Decision Tree
- **Moshiour Rahman Prince** - Environment Model (UML Diagram, UPPAAL model, python code)
  Leader Selection: Logistic Regression

10 Declaration of Originality

We, herewith, declare that we have composed the present paper and work by ourselves and without using anything other than the cited sources and aids. Sentences or parts of sentences quoted are marked as such; full details of the publications concerned indicate other references about the statement and scope. The paper and work in the same or similar form have not been submitted to any examination body or published. This paper has not yet, even in part, been used in another examination or as a course performance. We agree that a plagiarism checker may check our work.

Date&Place - Shihab Ud Doula, Jasmeet Singh Matta, Moshiur Rahman Prince

Bibliography

[BDL04] Behrmann, Gerd; David, Alexandre; Larsen, Kim Guldstrand: A tutorial on UPPAAL. In: Formal Methods for the Design of Real-Time Systems, pp. 200–236. Springer, 2004.

[BY03] Bengtsson, Johan; Yi, Wang: Timed automata: Semantics, algorithms and tools. In: Lectures on Concurrency and Petri Nets, pp. 87–124. Springer, 2003.

[IAH20] Iqbal, Sadaf; Arshad, Shazad; Hussain, Suhaib: Machine learning for autonomous driving: A survey. IEEE Access, 8:205546–205567, 2020.

[Ku11] Kunze, Rolf; Ramakers, Robbert; Henning, Klaus; Jeschke, Sabina: Organization and operation of electronically coupled truck platoons on German motorways. In: Proceedings of the 14th International IEEE Conference on Intelligent Transportation Systems (ITSC). IEEE, pp. 819–824, 2011.

[Pe11] Pedregosa, Fabian; Varoquaux, Gaël; Gramfort, Alexandre; Michel, Vincent; Thirion, Bertrand; Grisel, Olivier; Blondel, Mathieu; Prettenhofer, Peter; Weiss, Ron; Dubourg, Vincent et al.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[SZL16] Su, Yuchao; Zhang, Yanyan; Lv, Changle: Machine learning-based adaptive cruise control design considering car-following dynamics. IEEE Transactions on Vehicular Technology, 65(8):6566–6571, 2016.

[TJS16] Tsugawa, Sadayuki; Jeschke, Sabina; Shladover, Steven E: A review of truck platooning projects for energy savings. IEEE Transactions on Intelligent Vehicles, 1(1):68–77, 2016.

[Zh24] Zhang, Tianya Terry; Jin, Peter J; McQuade, Sean T; Bayen, Alexandre; Piccoli, Benedetto: Car-following models: A multidisciplinary review. IEEE Transactions on Intelligent Vehicles, 2024.