Embedded Systems Engineering

(Prof. Dr. Stefan Henkler)

Truck Platooning

Abhinandan Dinakar
Embedded System Engineering
Fachhochschule Dortmund
Dortmund, Germany

Nikhil Ganapathy Manjapura
Embedded System Engineering
Fachhochschule Dortmund
Dortmund, Germany

Shreya Manasali
Embedded System Engineering
Fachhochschule Dortmund
Dortmund, Germany

***Abstract* - Truck Platooning system is automation and cooperation system that leads a way for automated vehicle implementation. In this study we have considered platooning of lead truck with driver followed by other automated trucks. Addition to that we have considered many of the challenges regarding platooning system and possible effects and potential dangers in fully automated platooning system. We have discussed on what are the main modules that need to be considered for platooning implementation and the possible outcomes in case of failures. We conclude that each part of the system should be responsive to the stimuli of relevant reasons available..**

I. Introduction

Vehicle platooning is has the capability to reduce transportation cost and emissions [3]. Many companies are working on platooning technology as booming sensors and communication technologies makes this solution affordable and reachable. This technology has many advantages, such as, controllers reaction time is smaller than that of human driver. Also large quantity goods transportation can be achieved with less man power. But in real scenarios, as platooning is completely automated, system is more prone to errors. System failure in this technology can be fatal.

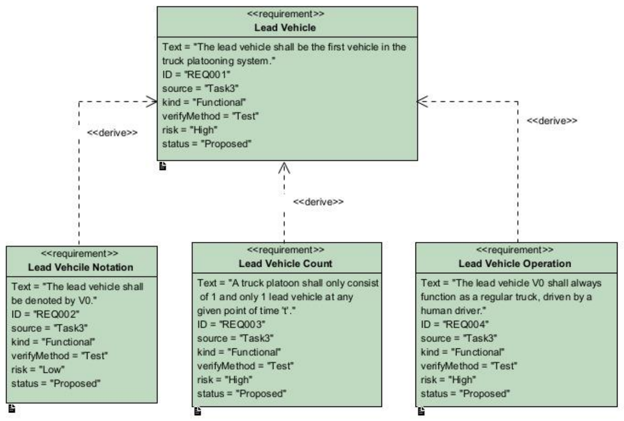
First system developed to control longitudinal dynamics is the cruise control(CC) [2]. Platooning trucks uses proximity sensors like Radar sensor which measures the distance from obstacles ahead and adjusts its velocity which led to the development of Adaptive Cruise Control(ACC) [1]. Vehicles adjust its velocity according to the preceeding vehicle.

Moreover, platoons can be classified on other two terminologies, namely homogenous and heterogenous. Homogenous refers to platoons in which vehicles have same dynamical capabilities. In contrast, Heterogenous refers to platoons with vehicles that does not share identical dynamic capabilities. Here we concentrate on longitudinal models where lead truck is followed by follower trucks in single line manner.

II. Requirements

It's the science and discipline concerned with analyzing requirements [4]. There are 5 main elements to derive System specifications and they are Requirement Elicitation, Requirement Analysis, Requirement Specification, Requirement Validation, Requirement Management. A Software Requirement Specification document will be created at the initial stage which documents the requirements for a system or component. Typically included are functional requirements, performance requirements, interface requirements, design requirements, and development standards.

A requirement diagram offers a graphical representation of the requirements. The properties which requirements should cover are priority, source, risk, status and verification method.



<<requirement>>
Lead Vehicle
Text = "The lead vehicle shall be the first vehicle in the truck platooning system."
ID = "REQ001"
source = "Task3"
kind = "Functional"
verifyMethod = "Test"
risk = "High"
status = "Proposed"

<<requirement>>
Lead Vehcile Notation
Text = "The lead vehicle shall be denoted by V0."
ID = "REQ002"
source = "Task3"
kind = "Functional"
verifyMethod = "Test"
risk = "Low"
status = "Proposed"

<<requirement>>
Lead Vehicle Count
Text = "A truck platoon shall only consist of 1 and only 1 lead vehicle at any given point of time 't'."
ID = "REQ003"
source = "Task3"
kind = "Functional"
verifyMethod = "Test"
risk = "High"
status = "Proposed"

<<requirement>>
Lead Vehicle Operation
Text = "The lead vehicle V0 shall always function as a regular truck, driven by a human driver."
ID = "REQ004"
source = "Task3"
kind = "Functional"
verifyMethod = "Test"
risk = "High"
status = "Proposed"

Figure 1. Requirement Diagram for Lead Vehicle in Truck Platooning

In Fig. 1, the Functional requirements related to Lead Vehicle are captured. The symbol <<derive>> explains that there are 3 subsystem requirements derived System requirement. The System Requirement illustrates that the Lead Vehicle shall be the first Vehicle in the Truck Platooning System with its requirement properties.

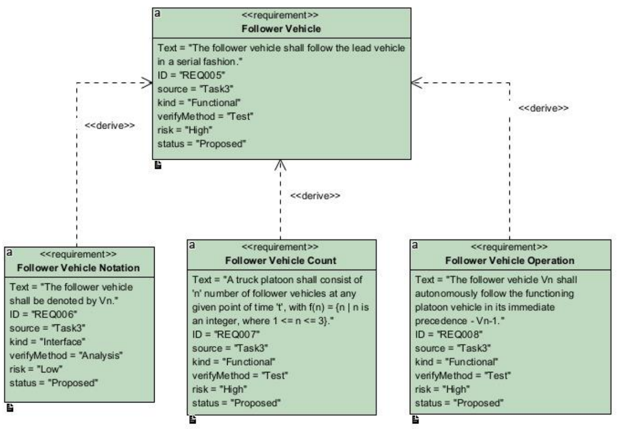


Figure 2. Requirement Diagram for Follower Vehicle in Truck Platooning

The graphical representation of Follower Vehicle requirements can be seen in Fig. 2. Also there are 3 subsystem requirements are derived from System Requirement. With the System Requirement as “The Follower Vehicle shall follow the Lead Vehicle in a Serial fashion”, a subsystem requirement can be analyzed as “The Follower Vehicle Vn shall autonomously follow the functioning Platoon vehicle in its immediate precedence V_{n-1}.

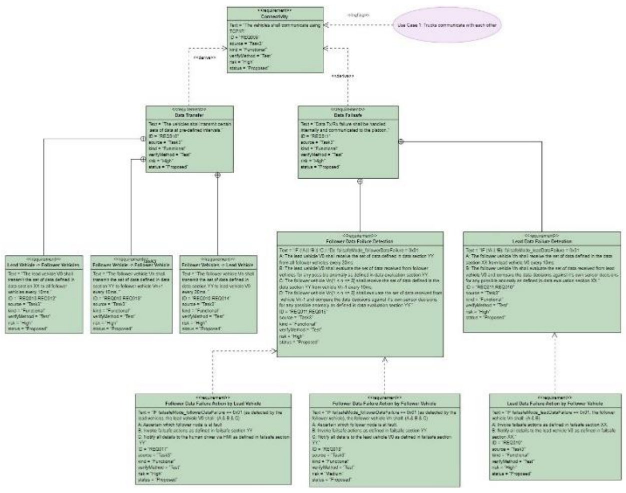


Figure 3. Requirement Diagram for Connectivity in Truck Platooning

The above Fig. 3 shows the Requirement diagram for Connectivity in Truck Platooning. Also the Containment(⊗) relation can be seen in the above Fig 3. The Containment relation makes it possible to deconstruct a composite requirement into several single requirements, which are then easier to trace with regard to architecture and tests[4]. There are 2 requirements “Data Transfer” and “Data FailSafe”, which are further deconstructed into several single requirements.

III. Model Section

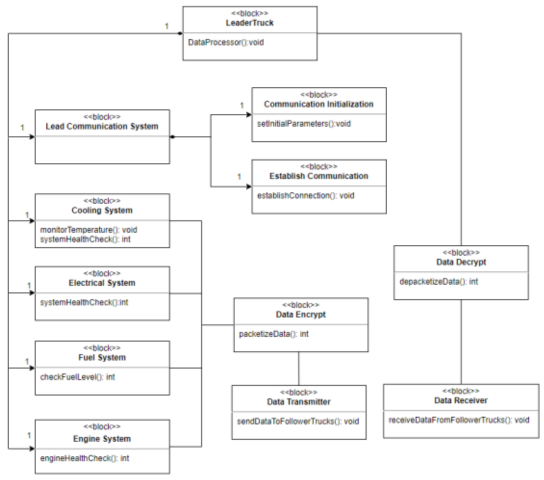


Figure 4. Block Diagram - Lead Truck

Fig. 4 and Fig. 5 represents the Block Diagrams of Lead and Follower Truck Platooning System modules that are taken into consideration in this study.

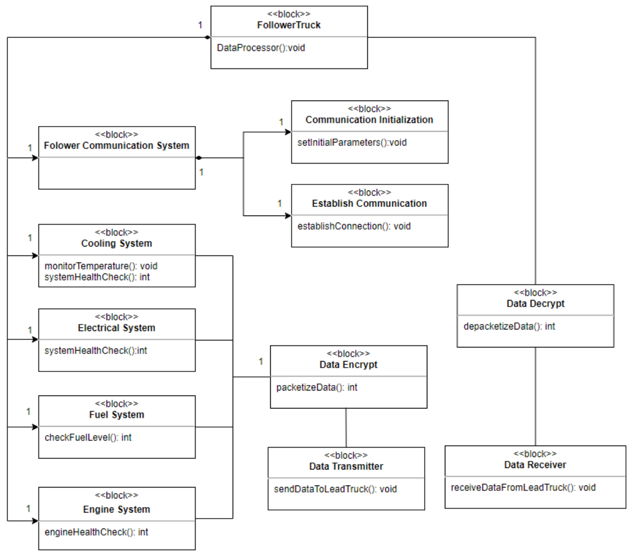


Figure 5. Block Diagram - Follower Truck

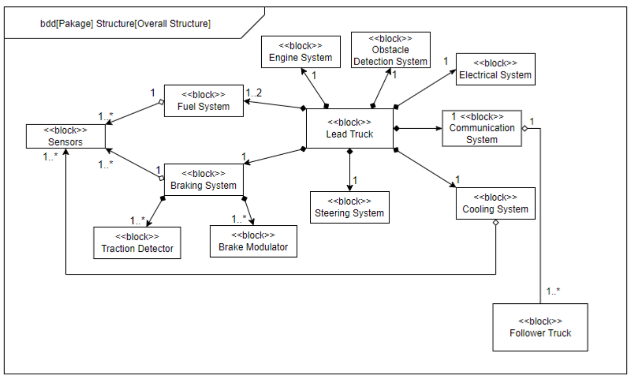


Figure 6. Block Definition Diagram

Block Definition Diagram (BDD) represents the main system modules that need to be considered in platooning system. BDD depicts the block dependencies on other block and gives an overview of this system. Each block in BDD is explained in Internal Block Diagram (IBD).

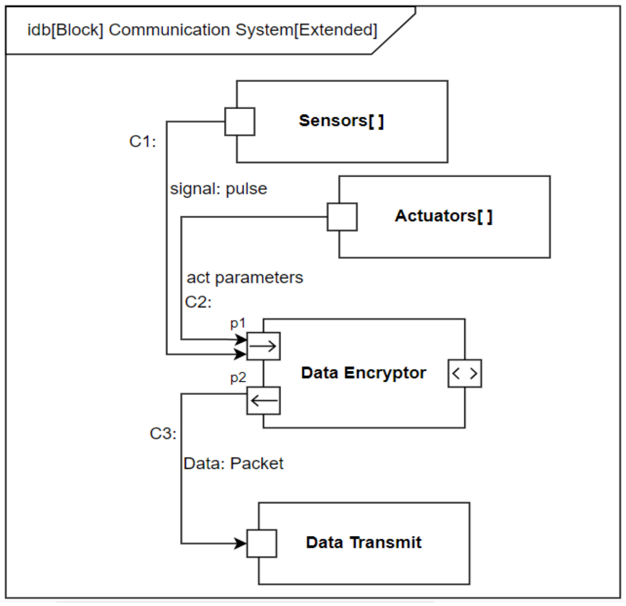


Figure 7. IBD - Communication System

Fig. 7 illustrates Internal Block Diagram of Communication System in which sensor and actuator data are communicated from one truck to another. Data is encrypted and decrypted at transmitter and receiver respectively.

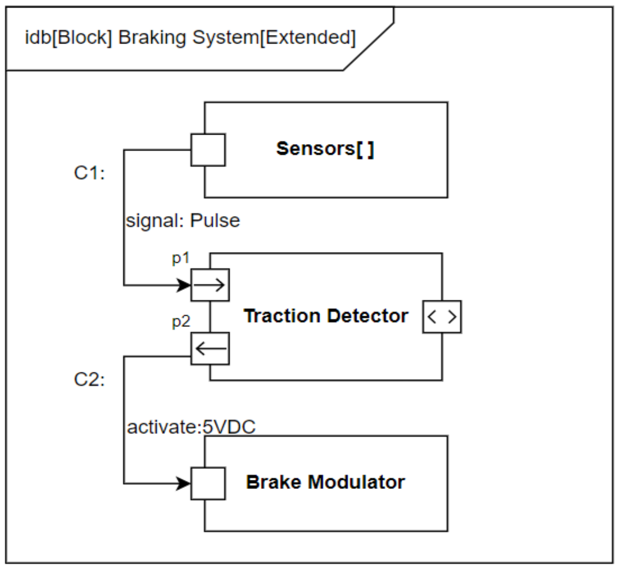


Figure 8. IBD - Brake System

Fig. 8 represents Internal Block Diagram of Braking System with sensors, traction detector and brake modulator as internal modules. Sensor output will be the input to traction detector block which in turn process the data and send activation signal to the Brake modulator.

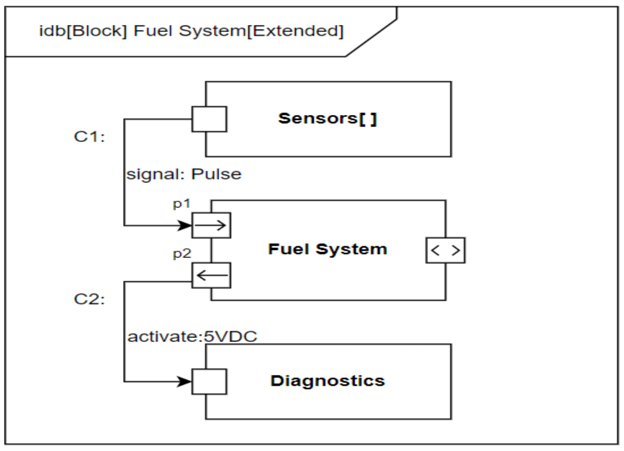


Figure 9. IBD - Fuel System

Fig. 9 represents the Fuel System, in which fuel level is checked in all truck and the same shall be indicated to the driver in the lead truck.

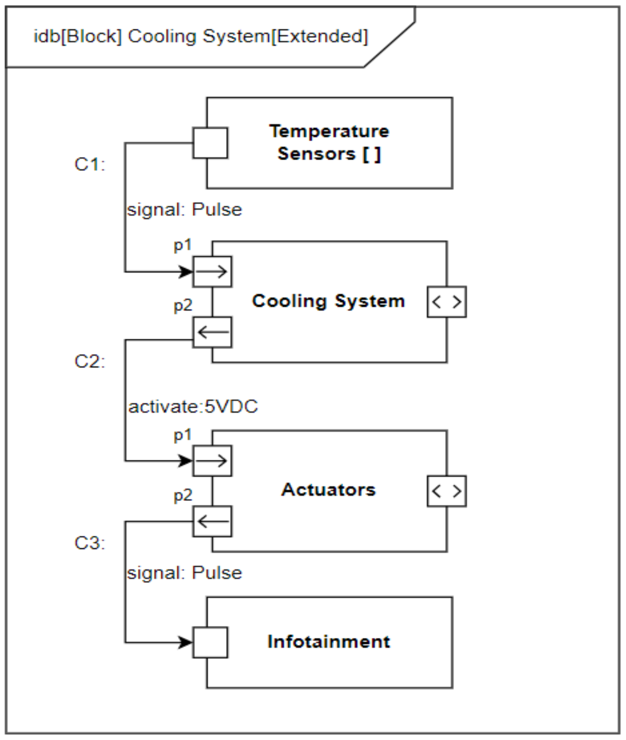


Figure 10. IBD - Cooling System

Fig. 10 represents Internal Block Diagram of Cooling System with sensors and actuators. If there is variation in temperature beyond threshold, actuators like fan and coolant pumps shall be triggered and the same shall be indicated in the dashboard.

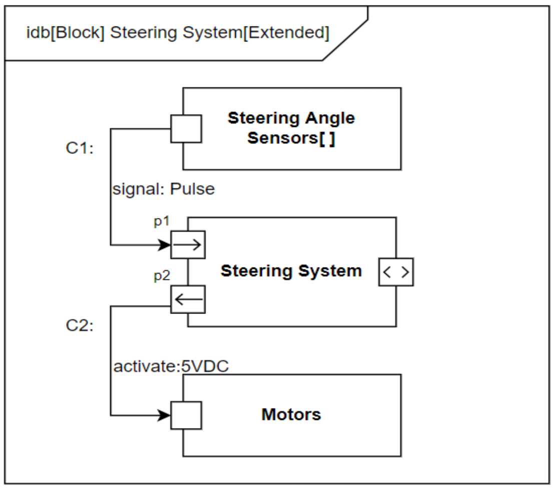

idb[Block] Steering System[Extended]
Steering Angle Sensors[ ]
C1:
signal: Pulse
p1
Steering System
p2
C2:
activate:5VDC
Motors

Figure 11. Steering System

Fig. 11 represents the Internal Block Diagram of Steering System. This system shall get the steering angle and be monitored and plan the direction accordingly with lead truck.

IV. ALLOCATION DIAGRAM

Fig. 12 represents the Allocation Diagram which describes the design decision that assigns responsibility for meeting a requirement or implementing a behavior to structural elements of the System [4].

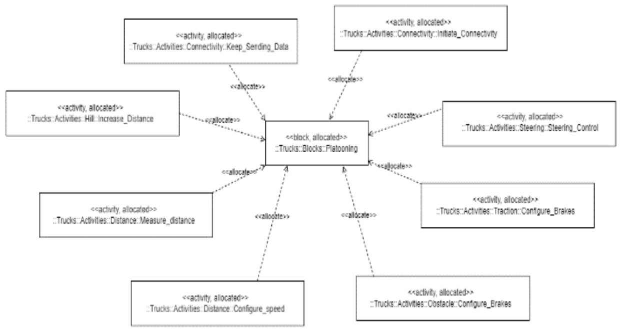

<<activity, allocated>>
::Trucks::Activities::Connectivity::Keep_Sending_Data
<<activity, allocated>>
::Trucks::Activities::Connectivity::Initiate_Connectivity
<<activity, allocated>>
::Trucks::Activities::Hill::Increase_Distance
<<block, allocated>>
::Trucks::Blocks::Platooning
<<activity, allocated>>
::Trucks::Activities::Steering::Steering_Control
<<activity, allocated>>
::Trucks::Activities::Distance::Measure_distance
<<activity, allocated>>
::Trucks::Activities::Traction::Configure_Brakes
<<activity, allocated>>
::Trucks::Activities::Distance::Configure_speed
<<activity, allocated>>
::Trucks::Activities::Obstacle::Configure_Brakes
<<allocate>>

Figure 12. Allocation Diagram for Truck Platooning

The Activities are segregated as per the functional requirements. Various blocks are included in this diagram explains different behaviors of the System. The Connectivity Blocks are responsible for Sending and Receiving Data, Hill Block for increasing distance, Distance Block for maintaining equal distance and Configuring speed, Obstacle and Traction Block for brakes activation and Steering Block for Steering Control.

V. SEQUENCE DIAGRAM

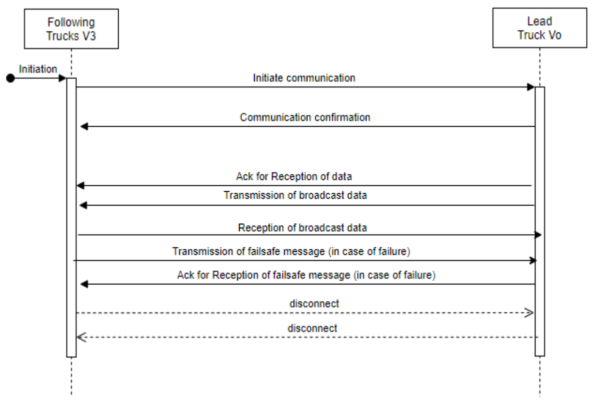

Following Trucks V3
Lead Truck Vo
Initiation
Initiate communication
Communication confirmation
Ack for Reception of data
Transmission of broadcast data
Reception of broadcast data
Transmission of failsafe message (in case of failure)
Ack for Reception of failsafe message (in case of failure)
disconnect
disconnect

Figure 13. Sequence Diagram

Fig. 13 depicts the communication sequence between lead and follower trucks from initiation to data packets exchange between lead and follower trucks. Here we have used TCP/IP as communication protocol between trucks.

VI. PARAMETRIC DIAGRAM

It is a specialization of an Internal Block Diagram that enforces constraints across the internal part value properties bound by the constraint block parameters. It is used to know dependencies between the parameters[4].

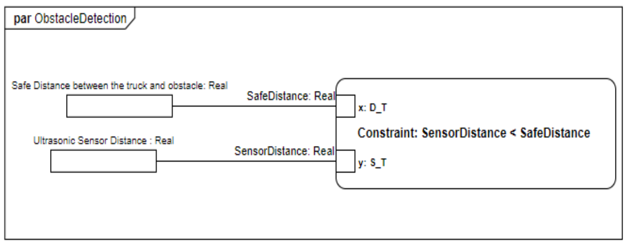

par ObstacleDetection
Safe Distance between the truck and obstacle: Real
SafeDistance: Real
x: D_T
Constraint: SensorDistance < SafeDistance
Ultrasonic Sensor Distance : Real
SensorDistance: Real
y: S_T

Figure 14. Obstacle detection paramteric diagram

Fig. 14 represents parametric diagram for obstacle detection module. The constraint for this system is to get activated only if the distance between the truck and the obstacle(SensorDistance) is less than the configured safe distance.

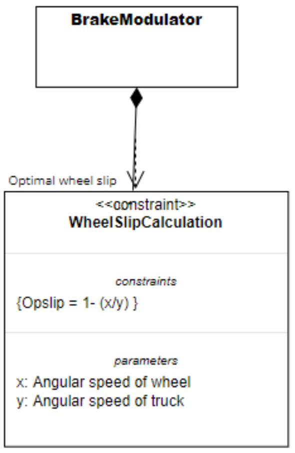


Figure 15. Brake modulator parametric diagram

Fig. 15 represents parametric diagram for Brake modulator module. The constraint for this system is to calculate the slip of truck wheels and to bring it to the configured optimal value in case of abnormality.

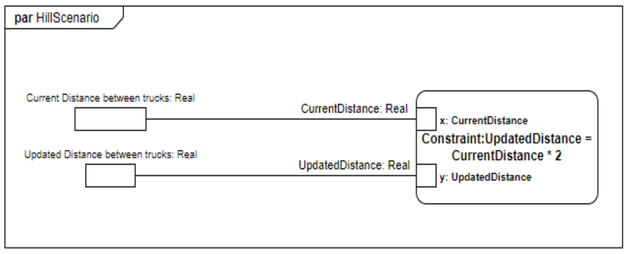

par HillScenario
Current Distance between trucks: Real
CurrentDistance: Real
x: CurrentDistance
Constraint:UpdatedDistance = CurrentDistance * 2
Updated Distance between trucks: Real
UpdatedDistance: Real
y: UpdatedDistance

Figure 16. Hill assistance parametric diagram

Fig. 16 represents parametric diagram for Hill assistance system. The constraint for this system is to update the distance between the trucks to twice of their current distance, when the trucks drive on a hill. This would help in avoiding mishap and maintaining safe distance between the trucks.

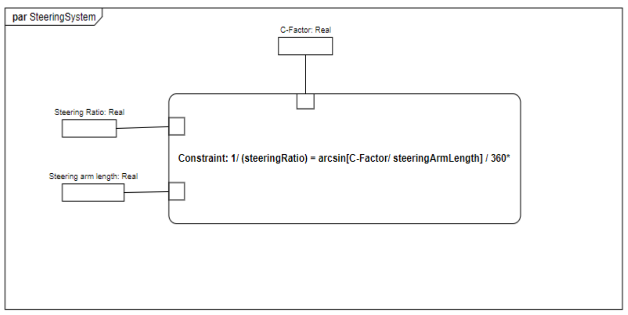

par SteeringSystem
C-Factor: Real
Steering Ratio: Real
Steering arm length: Real
Constraint: 1/ (steeringRatio) = arcsin[C-Factor/ steeringArmLength] / 360°

Figure 17. Steering control system parametric diagram

Fig. 17 represents parametric diagram for Steering control system. The constraint for this system is to maintain steering direction of the trucks with the help of calculated steering ratio.

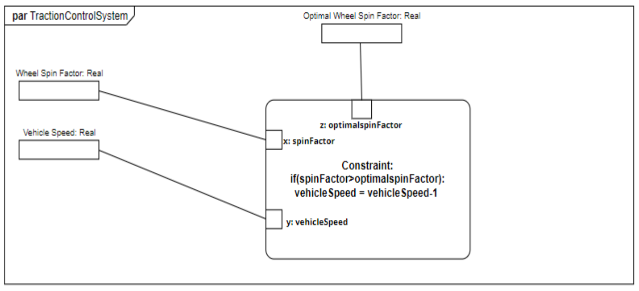

par TractionControlSystem
Optimal Wheel Spin Factor: Real
Wheel Spin Factor: Real
Vehicle Speed: Real
z: optimalspinFactor
x: spinFactor
Constraint:
if(spinFactor>optimalspinFactor):
vehicleSpeed = vehicleSpeed-1
y: vehicleSpeed

Figure 18. Traction Control System parametric diagram

Fig. 18 represents parametric diagram for Traction control system. The constraint for this system is to maintain spinning factor of the truck's wheel to an optimal configured wheel spin factor.

VII. ACTIVITY DIAGRAM

Activity Diagram specifies the sequential and concurrent behaviors that are connected by control flows and object flows [4]. The below Fig. 14 shows the Activity Diagram for Connectivity in Truck Platooning. This type of diagrams are used to specify the Functional Behavior of a System.

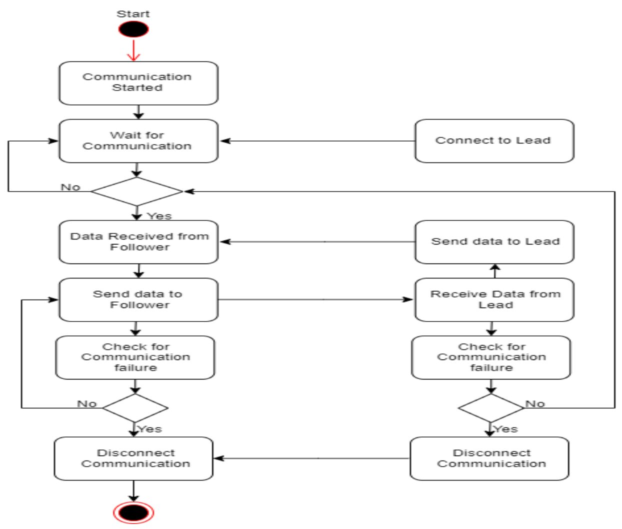


Figure 19. Activity Diagram for Connectivity in Truck Platooning

The Functional behavior for Connectivity can be seen in the above diagram. The Action Blocks with respect to the Control flow explain how the Communication will be established between Master and Slave Truck during Platoon stage. If communication failure happens then further steps will be decided by the Decision Block.

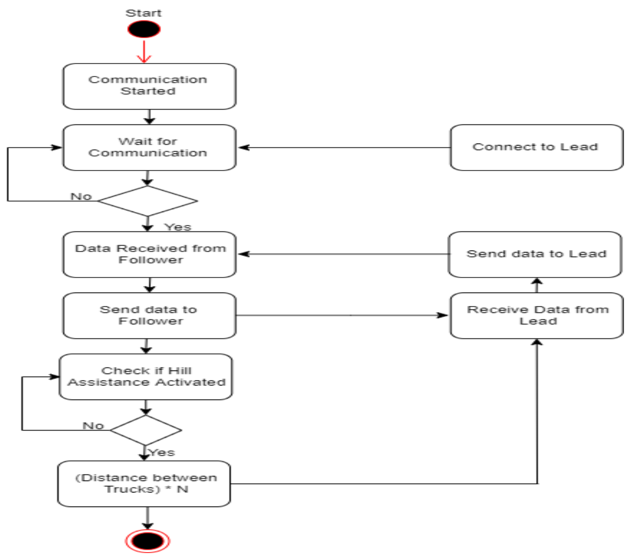


Figure 20. Activity Diagram for Hill Assist in Truck Platooning

The above Fig. 20 represents Activities carried out during Hill Assist Scenario in Truck Platooning. The Action Block will check whether the Hill Assist is Activated. Depending on that, decision block will activate next Action, which is increasing the Distance between the Master and Slave Trucks by configured value N.

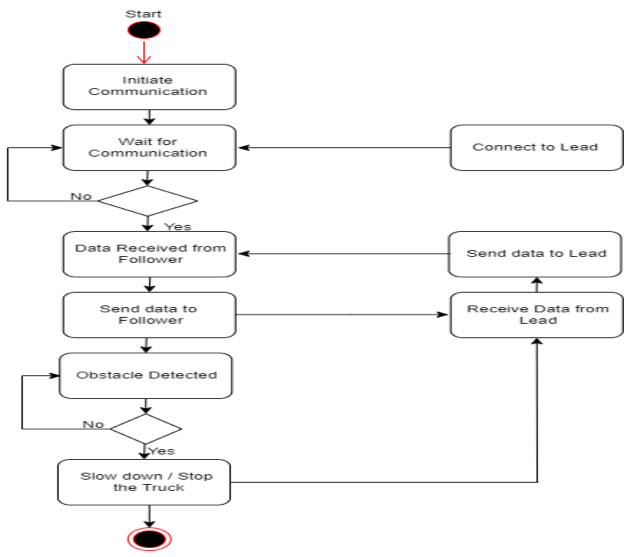


Figure 21. Activity Diagram for Obstacle Detection in Truck Platooning

The above Fig. 21 illustrates Activity flow during Obstacle Detection. When an Obstacle is detected, the Master or Slave Trucks will Slow down or Stop.

VIII. STATE MACHINE

A state machine is a behaviour model. It is used to model a particular discrete behaviour through finite state transition system in terms of transitions and states[4].

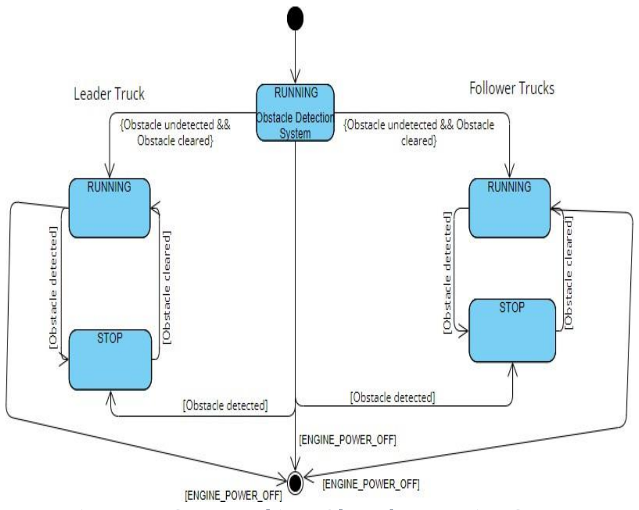


Figure 22. State machine -Obstacle Detection System

Fig. 22 represents the state machine diagram of obstacle detection system. This system shall monitor and notify the lead and following trucks about the obstacles if detected in their pathway.

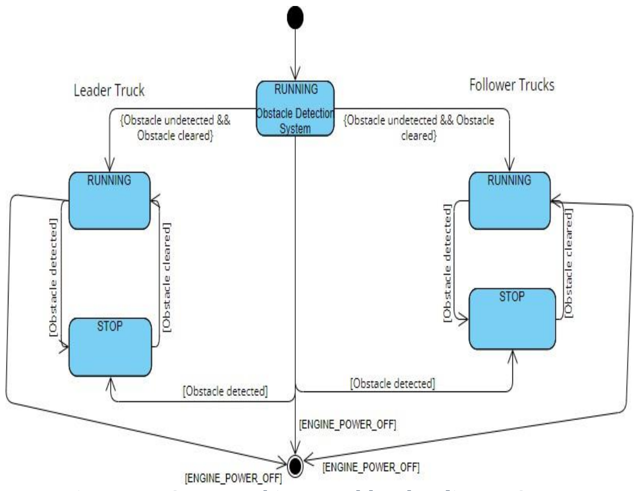


Figure 23. State machine -Fuel level Indicator System

Fig. 23 represents the state machine diagram of fuel level indicator system. This system shall notify lead truck driver about the fuel level in the fuel tanks of the trucks.

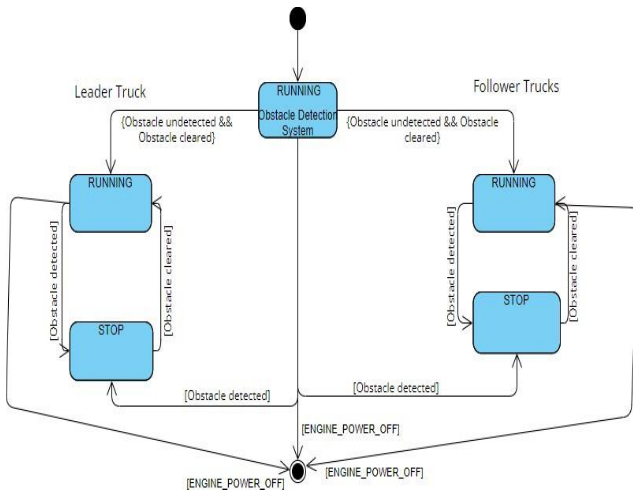


Figure 24. State machine -Traction Control System

Fig. 24 represents the state machine diagram of Traction Control system. This system shall slow down the trucks in case of high spinning of wheels or slippery road surface.

IX. IMPLEMENTATION

Main System that needs to be considered for longitudinal module is maintaining equidistance and obstacle detection system. Obstacle can be in front of lead truck or between the platoons. All the trucks are expected to detect the obstacle and act according to the situation.
Obstacle range information can be obtained from Lidar (laser ranging), sonar (sound ranging) or vision based (video and image processing) techniques. In this study we are taking proximity sensors in the truck into consideration for detecting the obstacle or incoming traffic. We have taken Arduino as the GPU and Ultrasonic sensor as proximity sensor. When the obstacle is detected by lead truck in certain distance, farther than critical distance, it indicates an alert signal to lead driver and the signal to stop or slowdown will be transmitted to follower trucks. Likewise, when obstacle is sensed by follower truck, it slows down or stop and send information to lead truck and other following trucks behind it.

Image above depicts the two scenarios of obstacle detection. Firstly, obstacle in critical distance which alerts the system with stop signal and in another scenario, obstacle in not so critical distance which send just alert signal. Obstacle detection algorithm includes polling of sensor input to the GPU. Our approach is reflexive based on the instantaneous local perception of obstacle position. This system prioritizes the obstacle based on their distance from the truck and take actions to avoid obstacles and crash. Another system that is implemented in this study is fuel level monitoring and indication. All the trucks fuel level is measured and the status is indicated to lead truck.

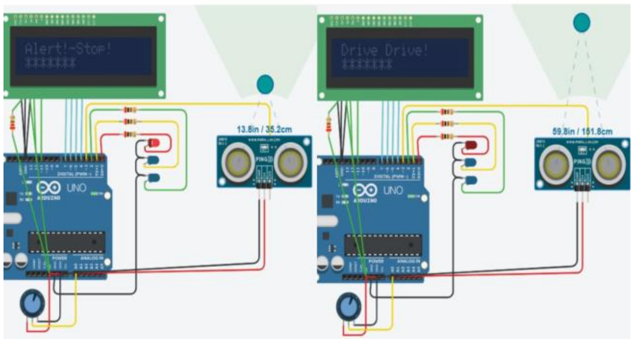
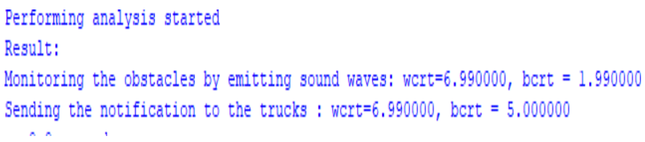


Figure 25: Module Implementation

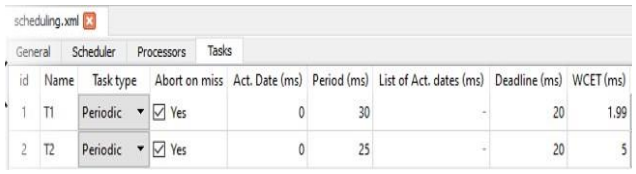
X. SCHEDULING

Scheduling is an action of assigning resources/processors to perform tasks within given constraints. We have scheduled tasks with respect to Obstacle Detection System using Static Priority Pre-emptive scheduler[5].


```
Performing analysis started
Result:
Monitoring the obstacles by emitting sound waves: wcrt=6.990000, bcrt = 1.990000
Sending the notification to the trucks : wcrt=6.990000, bcrt = 5.000000
```
Figure 26. Response time using pycpa

The defined tasks for this system are: T1: Monitoring the obstacles by emitting sound waves and T2: Sending the notification (Drive/Stop) to the trucks. After scheduling the tasks using pycpa we could derive the results shown in Fig. 26. Tasks T1 and T2 are assigned to a single processor.

There are various algorithms used for scheduling of the tasks. Earliest deadline first is one such algorithm which schedules the task with least deadline first. Rate monotonic is another algorithm which is a static priority assigning algorithm. It schedules the task with least time period first.


scheduling.xml

General | Scheduler | Processors | Tasks

id	Name	Task type	Abort on miss	Act. Date (ms)	Period (ms)	List of Act. dates (ms)	Deadline (ms)	WCET (ms)
1	T1	Periodic	☑ Yes	0	30	-	20	1.99
2	T2	Periodic	☑ Yes	0	25	-	20	5

Figure 27. EDF and RM Scheduling

In Fig. 27 time periods, deadlines and worst case execution time have been set for both the tasks using Simso tool.

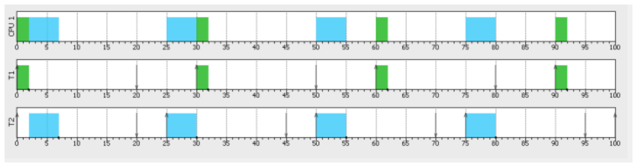


Figure 28. Timing diagram for EDF

Fig. 28 represents timing diagram for EDF algorithm. Both the tasks meet their deadlines and the tasks are schedulable as the utilization factor is 0.26 which is less than 1.

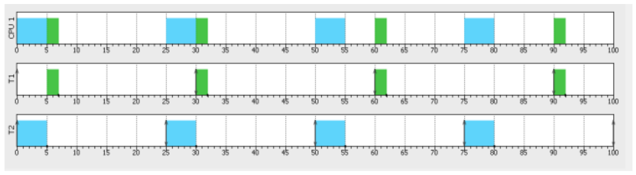


Figure 29. Timing diagram for RM

Fig. 29 represents timing diagram for RM algorithm. Both the tasks meet their deadlines and the tasks are schedulable as the utilization factor is 0.26 is less than 0.82 (least upper bound of the processor utilization factor).

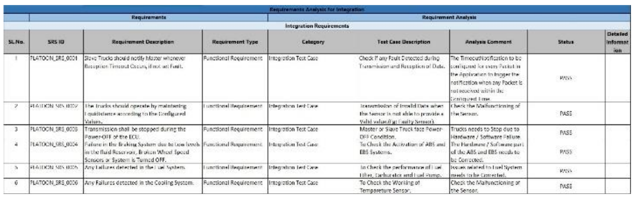
XI. TESTING

The main purpose of Testing is to determine failures and that can be corrected at the initial stages. Testing examines the behavior of the Software under test by Verification and Validation. It helps in analyzing and reviewing the product, improving the coding technique and many more. For testing, we use anonyms data as input to check the Performance of the System. The quality of the Software can be assured by mapping the Requirements to that Test Case.

The testing process starts with Designing the Test cases, preparing the Test Data, Running the Program with Test Data, comparing results with Test Cases and preparing the Test Report.

We are using the Google Test Framework (which is also known as gtest) is a unit testing library allowing unit-testing of C as well as C++ sources. We have created 6 Test Cases with respect to our Software Requirement Specifications (SRS). We will be checking some of the failures in the System. If the Test Cases passes then it indicates that there is a problem in that particular System and tells the Developer to look in to that Module.

Test Plan:
A Test Plan will be created before starting the Testing phase. A test plan will have Software Requirement Specification ID's, Requirement Description, Testing Category, Test Case Description, Status (Pass/Fail) and Analysis Comments.

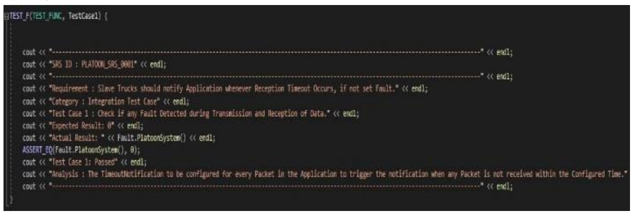


Requirements Analysis for Integration								
	Requirements			Requirement Analysis				
Integration Requirements								
SL.No.	SRS ID	Requirement Description	Requirement Type	Category	Test Case Description	Analysis Comment	Status	Detailed Information
1	PLATOON_SRS_0001	Slave Trucks should notify Master whenever Reception Timeout Occurs, if not set Fault.	Functional Requirement	Integration Test Case	Check if any Fault Detected during Transmission and Reception of Data.	The TimeoutNotification to be configured for every Packet in the Application to trigger the notification when any Packet is not received within the Configured Time.	PASS	
2	PLATOON_SRS_0002	The trucks should operate by maintaning Equidistance according to the Configured Values.	Functional Requirement	Integration Test Case	Transmission of Invalid Data when the Sensor is not able to provide a Valid value(Eg: Faulty Sensor).	Check the Malfunctioning of the Sensor.	PASS	
3	PLATOON_SRS_0003	Transmission shall be stopped during the Power-OFF of the ECU.	Functional Requirement	Integration Test Case	Master or Slave Truck face Power-OFF Condition.	Trucks needs to Stop due to Hardware / Software Failure.	PASS	
4	PLATOON_SRS_0004	Failure in the Braking System due to Low levels in the fluid Reservoir, Broken Wheel Speed Sensors or System is Turned OFF.	Functional Requirement	Integration Test Case	To Check the Activation of ABS and EBS Systems.	The Hardware / Software part of the ABS and EBS needs to be Corrected.	PASS	
5	PLATOON_SRS_0005	Any Failures detected in the Fuel System.	Functional Requirement	Integration Test Case	To Check the performance of Fuel Filter, Carburator and Fuel Pump.	Issues related to Fuel System needs to be Corrected.	PASS	
6	PLATOON_SRS_0006	Any Failures detected in the Cooling System.	Functional Requirement	Integration Test Case	To Check the Working of Tempareture Sensor.	Check the Malfunctioning of the Sensor.	PASS	

Figure 30. Test Plan for Truck Platooning

Requirement ID's will be mapped to those particular Test Cases and Testing will be done according to the Requirement and Test Case Description. The Test Logs will be names as per SRS ID's for the reference and if the Test Case fails then the justification will be recorded in the Detailed Information column.

Test Case 1:
The Master will be sending data to the Slaves with a Configured Periodicity. If the Slave missed any packet of Data, then the Time-Out occurs. Once this scenario happens then the Slave should notify Master about Time-Out.

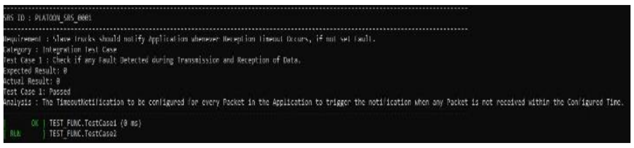


```
TEST_F(TEST_FUNC, TestCase1) {


    cout << "------------------------------------------------------------------------------------------------" << endl;
    cout << "SRS ID : PLATOON_SRS_0001" << endl;
    cout << "------------------------------------------------------------------------------------------------" << endl;
    cout << "Requirement : Slave Trucks should notify Application whenever Reception Timeout Occurs, if not set Fault." << endl;
    cout << "Category : Integration Test Case" << endl;
    cout << "Test Case 1 : Check if any Fault Detected during Transmission and Reception of Data." << endl;
    cout << "Expected Result: 0" << endl;
    cout << "Actual Result: " << Fault.PlatoonSystem() << endl;
    ASSERT_EQ(Fault.PlatoonSystem(), 0);
    cout << "Test Case 1: Passed" << endl;
    cout << "Analysis : The TimeoutNotification to be configured for every Packet in the Application to trigger the notification when any Packet is not received within the Configured Time."
    cout << "------------------------------------------------------------------------------------------------" << endl;
}
```

Figure 31. Slave to Notify Master in case of Reception TimeOut occurs.

Result 1:
The Slave trucks will notify the Master if any Data Packet is not received within the Configured time. The TimeoutNotification to be configured for every Packet in the Application to trigger the notification when any Packet is not received within the Configured Time.

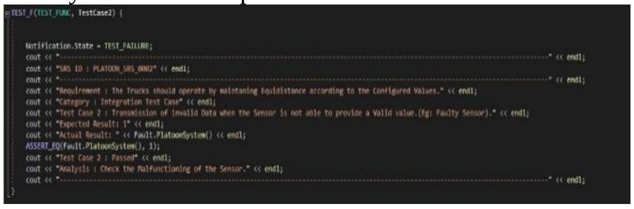


```
------------------------------------------------------------------------------------------------
SRS ID : PLATOON_SRS_0001
------------------------------------------------------------------------------------------------
Requirement : Slave Trucks should notify Application whenever Reception Timeout Occurs, if not set Fault.
Category : Integration Test Case
Test Case 1 : Check if any Fault Detected during Transmission and Reception of Data.
Expected Result: 0
Actual Result: 0
Test Case 1: Passed
Analysis : The TimeoutNotification to be configured for every Packet in the Application to trigger the notification when any Packet is not received within the Configured Time.

[       OK ] TEST_FUNC.TestCase1 (0 ms)
[ RUN      ] TEST_FUNC.TestCase2
```

Figure 32. Tested Result of Data Packet Reception.

Test Case 2:
The Trucks should operate by maintaining Equal Distance between them. This functionality completely depends on the values from Sensors of all the Trucks and Data received by the Master Truck. Here we will check whether all the Sensors are giving the proper Data. It also checks if there are any Fault Sensors present in the Trucks.

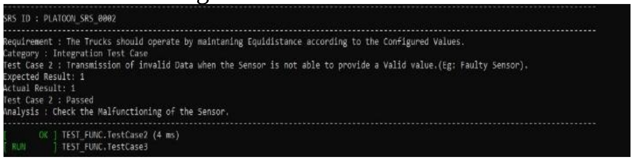


```
TEST_F(TEST_FUNC, TestCase2) {


    Notification.State = TEST_FAILURE;
    cout << "------------------------------------------------------------------------------------------------" << endl;
    cout << "SRS ID : PLATOON_SRS_0002" << endl;
    cout << "------------------------------------------------------------------------------------------------" << endl;
    cout << "Requirement : The Trucks should operate by maintaning Equidistance according to the Configured Values." << endl;
    cout << "Category : Integration Test Case" << endl;
    cout << "Test Case 2 : Transmission of invalid Data when the Sensor is not able to provide a Valid value.(Eg: Faulty Sensor)." << endl;
    cout << "Expected Result: 1" << endl;
    cout << "Actual Result: " << Fault.PlatoonSystem() << endl;
    ASSERT_EQ(Fault.PlatoonSystem(), 1);
    cout << "Test Case 2 : Passed" << endl;
    cout << "Analysis : Check the Malfunctioning of the Sensor." << endl;
    cout << "------------------------------------------------------------------------------------------------" << endl;
}
```

Figure 33. Malfunctioning of the Sensors

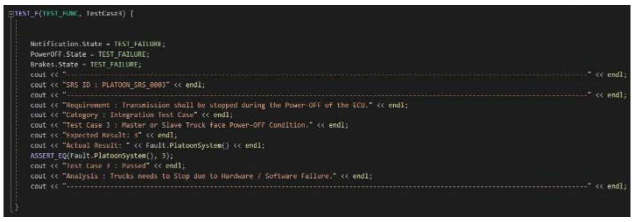
Result 2:
The Truck's sensor plays an important role to main equal distance between the Trucks. The Test Case will get passed when there is a faulty sensor present in the System. Check the Malfunctioning of the Sensor.



```
------------------------------------------------------------------------------------------------
SRS ID : PLATOON_SRS_0002
------------------------------------------------------------------------------------------------
Requirement : The Trucks should operate by maintaning Equidistance according to the Configured Values.
Category : Integration Test Case
Test Case 2 : Transmission of invalid Data when the Sensor is not able to provide a Valid value.(Eg: Faulty Sensor).
Expected Result: 1
Actual Result: 1
Test Case 2 : Passed
Analysis : Check the Malfunctioning of the Sensor.
------------------------------------------------------------------------------------------------
[       OK ] TEST_FUNC.TestCase2 (4 ms)
[ RUN      ] TEST_FUNC.TestCase3
```

Figure 34. Tested Result for any Faulty Sensor.

Test Case 3:
In any scenario, if the ECUs of Master and Slave Trucks gets Power-OFF, then the Slave Trucks should notify the Master or vice-versa about the Power-OFF condition.



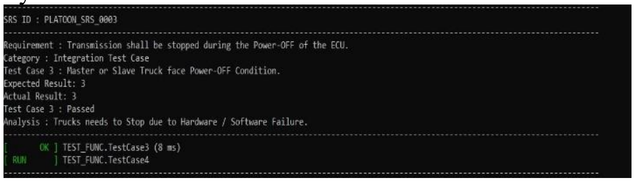
```
TEST_F(TEST_FUNC, TestCase3) {


    Notification.State = TEST_FAILURE;
    PowerOFF.State = TEST_FAILURE;
    Brakes.State = TEST_FAILURE;
    cout << "------------------------------------------------------------------------------------------------" << endl;
    cout << "SRS ID : PLATOON_SRS_0003" << endl;
    cout << "------------------------------------------------------------------------------------------------" << endl;
    cout << "Requirement : Transmission shall be stopped during the Power-OFF of the ECU." << endl;
    cout << "Category : Integration Test Case" << endl;
    cout << "Test Case 3 : Master or Slave Truck face Power-OFF Condition." << endl;
    cout << "Expected Result: 3" << endl;
    cout << "Actual Result: " << Fault.PlatoonSystem() << endl;
    ASSERT_EQ(Fault.PlatoonSystem(), 3);
    cout << "Test Case 3 : Passed" << endl;
    cout << "Analysis : Trucks needs to Stop due to Hardware / Software Failure." << endl;
    cout << "------------------------------------------------------------------------------------------------" << endl;

}
```

Figure 35. Power-Off condition in Master and Slave Trucks.

Result 3:
When Power-Off condition happens then the Trucks should stop and notify other Trucks. The Notification should be like "Hardware / Software Problem observed in the System".

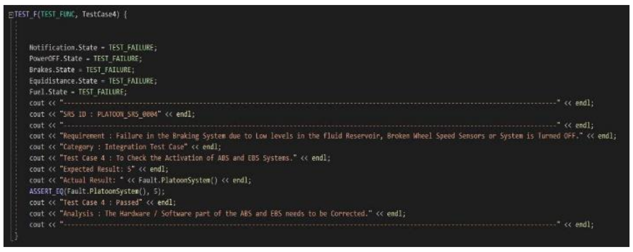


```
------------------------------------------------------------------------------------------------
SRS ID : PLATOON_SRS_0003
------------------------------------------------------------------------------------------------
Requirement : Transmission shall be stopped during the Power-OFF of the ECU.
Category : Integration Test Case
Test Case 3 : Master or Slave Truck face Power-OFF Condition.
Expected Result: 3
Actual Result: 3
Test Case 3 : Passed
Analysis : Trucks needs to Stop due to Hardware / Software Failure.
------------------------------------------------------------------------------------------------
[       OK ] TEST_FUNC.TestCase3 (8 ms)
[ RUN      ] TEST_FUNC.TestCase4
------------------------------------------------------------------------------------------------
```

Figure 36. Tested Result for Power-Off conditions.

Test Case 4:
Braking System is one of the main components in the Vehicle System. This test case is to check if there is any

Failure in the Braking System due to Low levels in the fluid Reservoir, Broken Wheel Speed Sensors or System is Turned OFF.

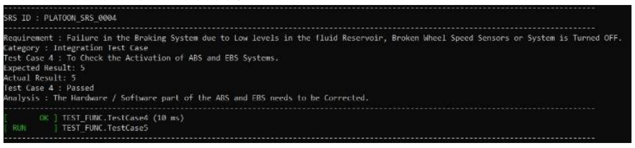


```
TEST_F(TEST_FUNC, TestCase4) {


    Notification.State = TEST_FAILURE;
    PowerOFF.State = TEST_FAILURE;
    Brakes.State = TEST_FAILURE;
    Equidistance.State = TEST_FAILURE;
    Fuel.State = TEST_FAILURE;
    cout << "------------------------------------------------------------------------------------------------" << endl;
    cout << "SRS ID : PLATOON_SRS_0004" << endl;
    cout << "------------------------------------------------------------------------------------------------" << endl;
    cout << "Requirement : Failure in the Braking System due to Low levels in the fluid Reservoir, Broken Wheel Speed Sensors or System is Turned OFF." << endl;
    cout << "Category : Integration Test Case" << endl;
    cout << "Test Case 4 : To Check the Activation of ABS and EBS Systems." << endl;
    cout << "Expected Result: 5" << endl;
    cout << "Actual Result: " << Fault.PlatoonSystem() << endl;
    ASSERT_EQ(Fault.PlatoonSystem(), 5);
    cout << "Test Case 4 : Passed" << endl;
    cout << "Analysis : The Hardware / Software part of the ABS and EBS needs to be Corrected." << endl;
    cout << "------------------------------------------------------------------------------------------------" << endl;
}
```

Figure 37. Failure in the Braking System

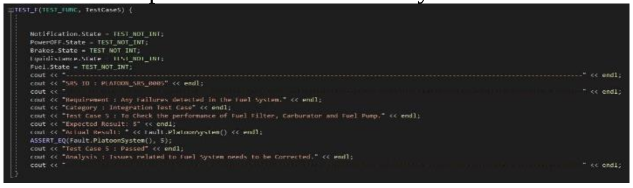
Result 4:
The Test Case gets passed when there is any problem observed in the Braking System due to Low levels in the fluid Reservoir, Broken Wheel Speed Sensors or System is Turned OFF. The Trucks should indicate the Drivers that there is a Hardware / Software part of the ABS and EBS needs to be Corrected.



```
------------------------------------------------------------------------------------------------
SRS ID : PLATOON_SRS_0004
------------------------------------------------------------------------------------------------
Requirement : Failure in the Braking System due to Low levels in the fluid Reservoir, Broken Wheel Speed Sensors or System is Turned OFF.
Category : Integration Test Case
Test Case 4 : To Check the Activation of ABS and EBS Systems.
Expected Result: 5
Actual Result: 5
Test Case 4 : Passed
Analysis : The Hardware / Software part of the ABS and EBS needs to be Corrected.
------------------------------------------------------------------------------------------------
[       OK ] TEST_FUNC.TestCase4 (10 ms)
[ RUN      ] TEST_FUNC.TestCase5
------------------------------------------------------------------------------------------------
```

Figure 38. Tested Result for Braking Failure.

Test Case 5:
This Test Case checks whether there is a problem occurred in the Fuel System. Also, to check the performance of Fuel Filter, Carburator and Fuel Pump. If there any problem detected in the Fuel System then it should indicate the Drivers to stop the Vehicle immediately.

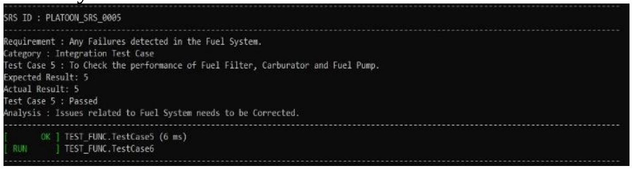


```
TEST_F(TEST_FUNC, TestCase5) {


    Notification.State = TEST_NOT_INT;
    PowerOFF.State = TEST_NOT_INT;
    Brakes.State = TEST_NOT_INT;
    Equidistance.State = TEST_NOT_INT;
    Fuel.State = TEST_NOT_INT;
    cout << "------------------------------------------------------------------------------------------------" << endl;
    cout << "SRS ID : PLATOON_SRS_0005" << endl;
    cout << "------------------------------------------------------------------------------------------------" << endl;
    cout << "Requirement : Any Failures detected in the Fuel System." << endl;
    cout << "Category : Integration Test Case" << endl;
    cout << "Test Case 5 : To Check the performance of Fuel Filter, Carburator and Fuel Pump." << endl;
    cout << "Expected Result: 5" << endl;
    cout << "Actual Result: " << Fault.PlatoonSystem() << endl;
    ASSERT_EQ(Fault.PlatoonSystem(), 5);
    cout << "Test Case 5 : Passed" << endl;
    cout << "Analysis : Issues related to Fuel System needs to be Corrected." << endl;
    cout << "------------------------------------------------------------------------------------------------" << endl;
}
```

Figure 39. Failure in the Fuel System

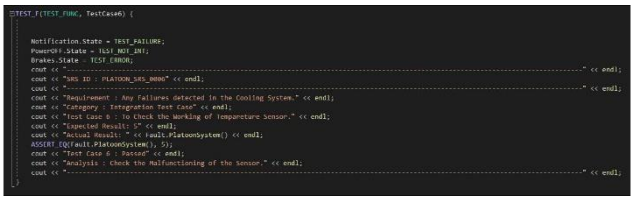
Result 5:
This is the indication that there is some problem in the Fuel Filter, Carburator and Fuel Pump. Issues related to those systems needs to be Corrected.



```
------------------------------------------------------------------------------------------------
SRS ID : PLATOON_SRS_0005
------------------------------------------------------------------------------------------------
Requirement : Any Failures detected in the Fuel System.
Category : Integration Test Case
Test Case 5 : To Check the performance of Fuel Filter, Carburator and Fuel Pump.
Expected Result: 5
Actual Result: 5
Test Case 5 : Passed
Analysis : Issues related to Fuel System needs to be Corrected.
------------------------------------------------------------------------------------------------
[       OK ] TEST_FUNC.TestCase5 (6 ms)
[ RUN      ] TEST_FUNC.TestCase6
------------------------------------------------------------------------------------------------
```

Figure 40. Tested Result for Fuel System Failure.

Test Case 6:
Test Case to check the working of Cooling System. In this Test, the operations of Tempareture Sensor will be analyzed.

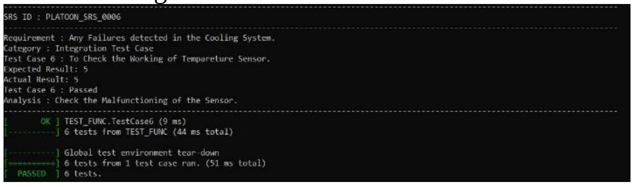


```
TEST_F(TEST_FUNC, TestCase6) {


    Notification.State = TEST_FAILURE;
    PowerOFF.State = TEST_NOT_INT;
    Brakes.State = TEST_ERROR;
    cout << "------------------------------------------------------------------------------------------------" << endl;
    cout << "SRS ID : PLATOON_SRS_0006" << endl;
    cout << "------------------------------------------------------------------------------------------------" << endl;
    cout << "Requirement : Any Failures detected in the Cooling System." << endl;
    cout << "Category : Integration Test Case" << endl;
    cout << "Test Case 6 : To Check the Working of Tempareture Sensor." << endl;
    cout << "Expected Result: 5" << endl;
    cout << "Actual Result: " << Fault.PlatoonSystem() << endl;
    ASSERT_EQ(Fault.PlatoonSystem(), 5);
    cout << "Test Case 6 : Passed" << endl;
    cout << "Analysis : Check the Malfunctioning of the Sensor." << endl;
    cout << "------------------------------------------------------------------------------------------------" << endl;
}
```

Figure 41. Failure in the Cooling System.

Result 6:
Positive result of this Test Case indicates that there is some problem in the Tempareture Sensor. Issues related to malfunctioning of the Sensor needs to be corrected.



```
------------------------------------------------------------------------------------------------
SRS ID : PLATOON_SRS_0006
------------------------------------------------------------------------------------------------
Requirement : Any Failures detected in the Cooling System.
Category : Integration Test Case
Test Case 6 : To Check the Working of Tempareture Sensor.
Expected Result: 5
Actual Result: 5
Test Case 6 : Passed
Analysis : Check the Malfunctioning of the Sensor.
------------------------------------------------------------------------------------------------
[       OK ] TEST_FUNC.TestCase6 (9 ms)
[----------] 6 tests from TEST_FUNC (44 ms total)

[----------] Global test environment tear-down
[==========] 6 tests from 1 test case ran. (51 ms total)
[  PASSED  ] 6 tests.
```

Figure 42. Tested Result for Cooling System Failure.

XII. REFERENCES

[1] Rajamani, R., 2015. Adaptive Cruise Control, Springer London, London, pp. 13–19. ISBN 978-1-4471-5058-9. doi: 10.1007/978-1-4471-5058-9_72.

[2] Teetor, R.R., 1950. “Speed control device for resisting operation of the accelerator”. URL https://www.google.com/patents/US2519859. US Patent 2,519,859.

[3] Tsugawa, S., Jeschke, S. and Shladover, S.E., 2016. “A review of truck platooning projects for energy savings”. IEEE Transactions on Intelligent Vehicles, Vol. 1, No. 1, pp. 68–77. doi:10.1109/TIV.2016.2577499.

[4] Lecture PPT (Prof. Dr. Stefan Henkler).

[5] Scheduling using pycpa URL: https://pycpa.readthedocs.io/en/latest/