HOCHSCHULE HAMM-LIPPSTADT

Autonomous Systems 1 Lab

SUBMITTED BY
ZAHID PARVEZ
DANISH ZAHEER MALIK
MUHAMMAD ROHAIL USMAN
QAMAR SAJJAD

Submitted to
Prof. Dr. Martin Hirsch
Prof. Dr. Stefan Henkler
Department Lippstadt 2
10.07.2022

Contents

List of Figures

ACKNOWLEDGEMENT

We would like to take this chance to express our sincere gratitude and respect to Hochschule Hamm-Lippstadt for giving us the means to complete our project work.

It gives us great pleasure and honour to express our sincere gratitude to our mentors, Prof. Dr. Martin Hirsch and Prof. Dr. Stefan Henkler, Department Lippstadt 2, for their invaluable guidance throughout this review.

We also want to express our gratitude to my family and friends for their unwavering moral support as we worked on our project.

Abstract

This study presents a review of theoretical and experimental work on truck platooning. A line of vehicles moving in close formation is referred to as a platoon. Only the first vehicle serves as the primary leading truck and is driven automatically while the following is assisted by the control system. This strategy's main goal is to reduce labour, time, fuel use, and greenhouse gas emissions. This is due to the fact that the trucks' close proximity in this arrangement enables more effective airflow to pass around them, which reduces overall energy consumption. The platoon configuration also enables increased road safety, transport capacity, driver comfort, reduced congestion, and personnel costs due to the absence of human intervention in the following vehicles. Three categories of academic research on truck platooning have been established: truck coordination, platoon formation maintenance, and fuel consumption in truck platoons. This paper presents the state of the art in each of these areas, classifies related publications according to their contributions, and discusses potential future research directions.

1 Motivation

The main concept of the project is to create an autonomous system in which several trucks can travel autonomously. The trucks should follow the main leading truck and follow the instructions given by the leading truck, such as reacting in accordance situation arises and make changes in speed and lateral. movements as per the demand of the system.

Initiating with defining the scenarios of the system, a lane-change phenomenon is implemented using python in pycharm and the leader election algorithm has been finalized using the KNN-algorithm.

2 Requirements of the System

To better understand the truck platoon system, we created UML diagrams in the requirement diagram, as shown in figure 1. The main requirement is that the system select a truck leader, and the slave trucks must follow. As we can see in the diagram, we have two more requirements: autonomous and connectivity. Connectivity means that the trucks must be connected, and autonomous means that the system should run autonomously. Furthermore, the system should detect any malfunctions. This could be any kind of failure. For example, we can check the tyre pressure. Another requirement is that the system perform all automotive functions, such as braking or giving indications, and to avoid collisions, the system should use sensors to avoid any kind of collision between them. If there is any kind of emergency braking system, the leader and all slave trucks should braking in any emergency situation.

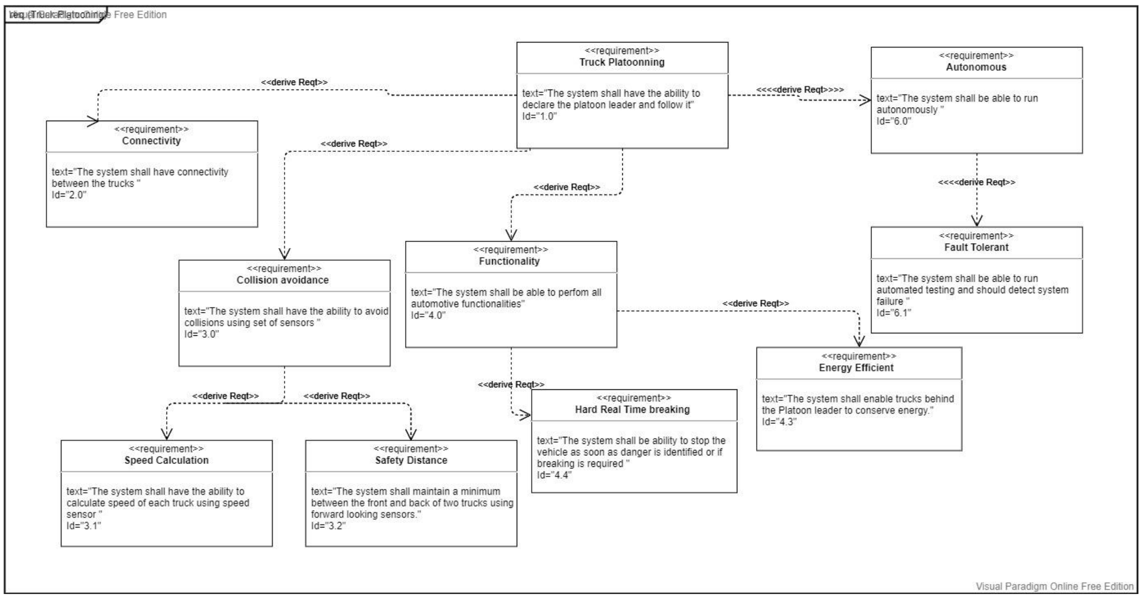

req [Truck Platooning]
Visual Paradigm Online Free Edition
<<requirement>>
Truck Platoonning
text="The system shall have the ability to declare the platoon leader and follow it"
Id="1.0"
<<derive Reqt>>
<<requirement>>
Connectivity
text="The system shall have connectivity between the trucks "
Id="2.0"
<<<<derive Reqt>>>>
<<requirement>>
Autonomous
text="The system shall be able to run autonomously "
Id="6.0"
<<derive Reqt>>
<<derive Reqt>>
<<<<derive Reqt>>
<<requirement>>
Fault Tolerant
text="The system shall be able to run automated testing and should detect system failure "
Id="6.1"
<<requirement>>
Collision avoidance
text="The system shall have the ability to avoid collisions using set of sensors "
Id="3.0"
<<requirement>>
Functionality
text="The system shall be able to perfom all automotive functionalities"
Id="4.0"
<<derive Reqt>>
<<requirement>>
Energy Efficient
text="The system shall enable trucks behind the Platoon leader to conserve energy."
Id="4.3"
<<derive Reqt>>
<<derive Reqt>>
<<derive Reqt>>
<<requirement>>
Hard Real Time breaking
text="The system shall be ability to stop the vehicle as soon as danger is identified or if breaking is required "
Id="4.4"
<<requirement>>
Speed Calculation
text="The system shall have the ability to calculate speed of each truck using speed sensor "
Id="3.1"
<<requirement>>
Safety Distance
text="The system shall maintain a minimum between the front and back of two trucks using forward looking sensors."
Id="3.2"
Visual Paradigm Online Free Edition

Figure 1: The Requirements of the System

3 Scenarios of the System

We have different scenarios in our system, and to understand it clearly, we have divided the scenarios into parts. In part one, we made a sequence diagram as shown in figure 2, which basically tells us the sequence of messages between objects. As we can see in the diagram, the master truck is sending messages to the cloud, and the cloud is requesting the truck ID for identification, and the master truck is sending the data back to the cloud.

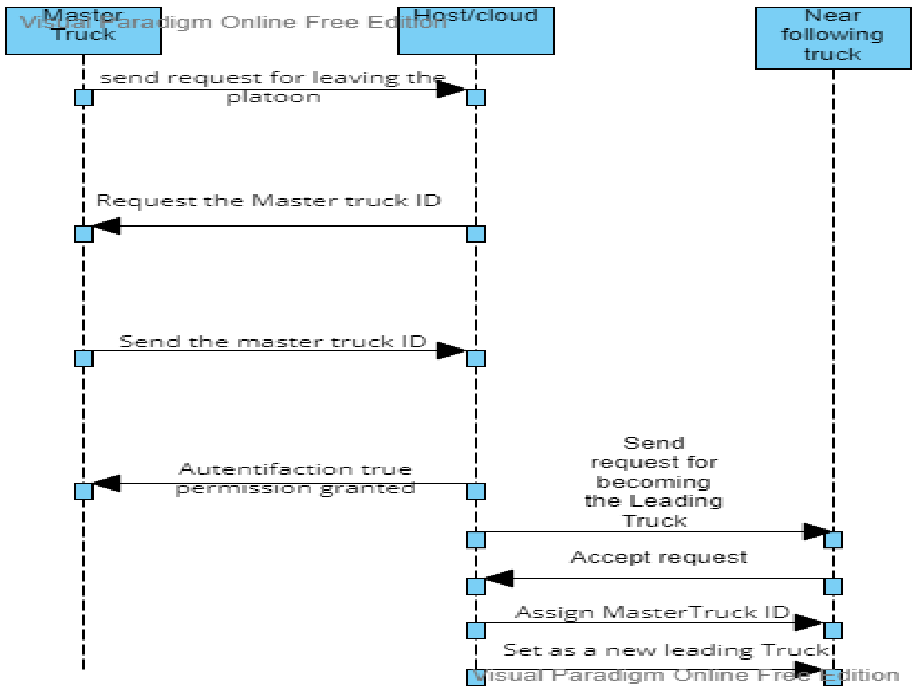

Master Truck
Visual Paradigm Online Free Edition
Host/cloud
Near following truck
send request for leaving the platoon
Request the Master truck ID
Send the master truck ID
Autentifaction true permission granted
Send request for becoming the Leading Truck
Accept request
Assign MasterTruck ID
Set as a new leading Truck
Visual Paradigm Online Free Edition

Figure 2: Scenario 1

3.1 Lane-Change Scenario

The idea for platoon lane-change scenario was generated having the necessity to either change the track in case of a hurdle ahead in the track or priority lane change demanded by the system for scenarios such a quitting the highway, etc. The sequence diagram for the system is as follow:

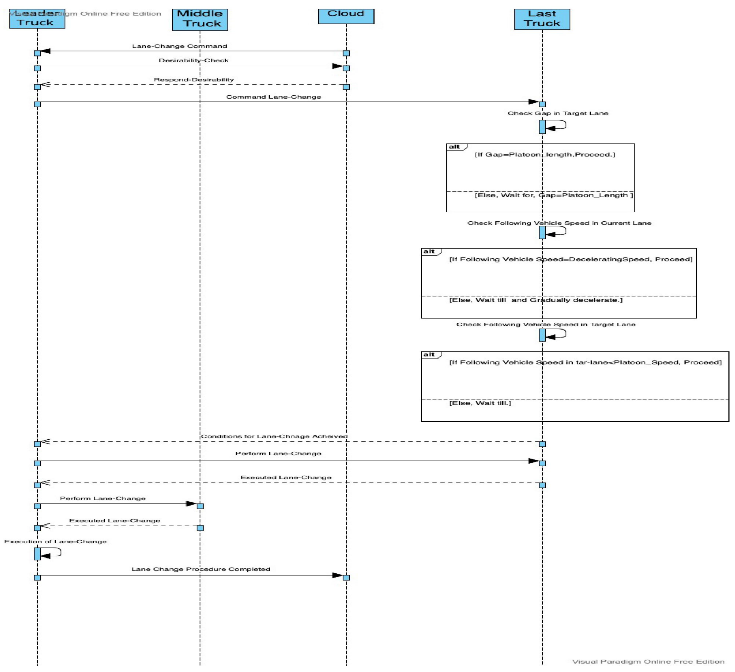

Leader Truck
Middle Truck
Cloud
Last Truck
Lane-Change Command
Desirability-Check
Respond-Desirability
Command Lane-Change
Check Gap in Target Lane
alt
[If Gap=Platoon_length,Proceed.]
[Else, Wait for, Gap=Platoon_Length]
Check Following Vehicle Speed in Current Lane
alt
[If Following Vehicle Speed=DeceleratingSpeed, Proceed]
[Else, Wait till and Gradually decelerate.]
Check Following Vehicle Speed in Target Lane
alt
[If Following Vehicle Speed in tar-lane<Platoon_Speed, Proceed]
[Else, Wait till.]
Conditions for Lane-Chnage Acheived
Perform Lane-Change
Executed Lane-Change
Perform Lane-Change
Executed Lane-Change
Execution of Lane-Change
Lane Change Procedure Completed
Visual Paradigm Online Free Edition

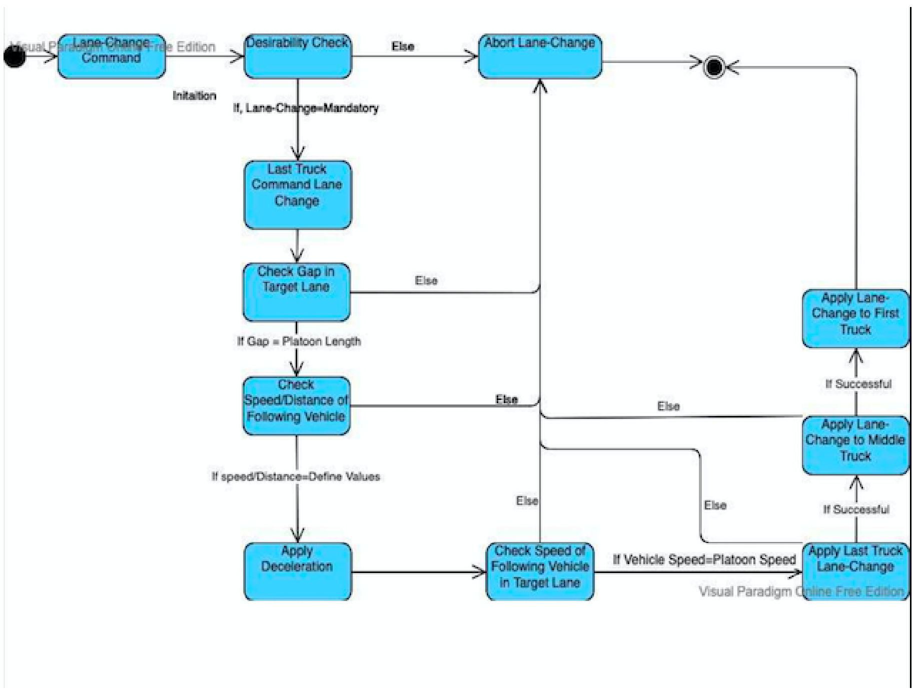
Figure 3: Scenario: Lane-Change (Sequence Diagram)

The lane-change scenario is first based on the desirability check and we have kept kept the systems limited to only change the lane when the priority is mandatory otherwise aborting the scenario is designed. Moreover, in order to make the system more robust, last vehicle first to change the lane is most suitable. For instance, as we have taken a platoon of three trucks, if the last vehicle changes the lane first, it will create easily possibilities for other preceding trucks to change the lane at the set speed because in that way only the last vehicle will have to go through the gap-creation process and the other preceding trucks will just implement the lane-change sequence. [4] [2]

Figure 3, which is uploaded in the Github repository [8], depicts that after the desirability check, the last truck checks the gap in the target lane and if the gap in the target lane matches the platoon length, it proceeds towards checking the speed of the following vehicles in the current lane in order to avoid the collision when switching to the pre-defined platoon lane-changing speed. Moreover, after decelerating the platoon for lane-change, the last vehicle then checks the speed of the following vehicle in the target lane in order to avoid collisions while changing the lane. If the speed of the following vehicle in the target lane matches or is lower than the speed of the platoon it applies the lane-change by first turning the blinkers on and then changing the lane from current lane to target lane.

To the support design workflow of the scenario and in order to ensure robustness and safety of the scenario, the state-machine diagram for the system is as below:

Figure 4, the state-diagram for lane-change scenario portrays how the system is directed to abort or wait state if the desired or pre-defined set of conditions are not meeting [2] [3] [4].



Lane-Change Command
Desirability Check
Else
Abort Lane-Change
Initaition
If, Lane-Change=Mandatory
Last Truck Command Lane Change
Check Gap in Target Lane
Else
If Gap = Platoon Length
Check Speed/Distance of Following Vehicle
Else
If speed/Distance=Define Values
Apply Deceleration
Else
Check Speed of Following Vehicle in Target Lane
If Vehicle Speed=Platoon Speed
Apply Last Truck Lane-Change
If Successful
Apply Lane-Change to Middle Truck
Else
Else
If Successful
Apply Lane-Change to First Truck
Visual Paradigm Online Free Edition

Figure 4: Scenario:Lane-Change (State-Machine Diagram)

3.2 Algorithmic Design

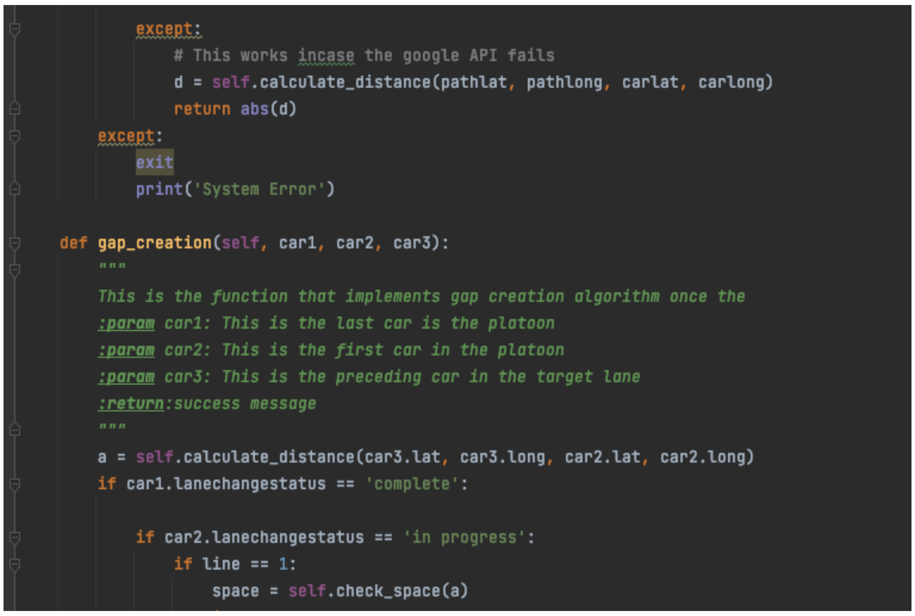
- Lane Change desire command to leader Truck.
- Desirability Check.
- If desirability=mandatory, Accept.
- Else Decline.
- Marking the current lane as i and the target lane as j.
- Command to last vehicle for Lane-Change.
- Retrieving speed of following vehicles in j and i via speed sensors.
- Retrieving distance of vehicles in j and i lane w.r.t platoon position via cameras.
- Selection of Gap for Last Truck: If gap in j $\neq$ platoon length.
- Decline and Wait.
- Repeat Step 9 until gap in j=platoon length.
- Check Speed of vehicles in current lane.
- If Decelerating Speed matches following vehicle speed, execute Deceleration.
- Else, continue and wait until Step 13 executes.
- Check Speed of Vehicles in Target Lane i.e., j.
- Check speed of following vehicle $<$ platoon speed.
- If speed of following vehicles in j $>$ platoon speed, Wait for Step 16=True.
- When Step 16=True, activate Lane-Change for Last Truck.
- Last Truck: Turn Blinkers on.

- Perform Lane Change.
- Command Last Truck successful lane-change.
- Middle Truck: Turn Blinkers on.
- Perform Lane Change.
- Command Middle Truck successful lane-change.
- Leader Truck: Turn Blinkers on.
- Perform Lane Change.
- Command Successful Platoon Lane-Change [8] [2].

3.3 Scenario Code Structuring using Python

To implement the lane-scenario, we initialized to code down the whole scenarios but as we were not able to do a run-time execution of the algorithm using the Carla Simulator due to unavailable system requirements, a proper simulation was not implemented. Moreover, we even tried to make our own environment and a few details of the coding version can be found in the GitHub in the folder, lane-change scenario. [8] [6]

Detailed commented file can be found in the GitHub Repository but a Code Snippets of the structure is as follow:



```
        except:
            # This works incase the google API fails
            d = self.calculate_distance(pathlat, pathlong, carlat, carlong)
            return abs(d)
    except:
        exit
        print('System Error')

def gap_creation(self, car1, car2, car3):
    """
    This is the function that implements gap creation algorithm once the
    :param car1: This is the last car is the platoon
    :param car2: This is the first car in the platoon
    :param car3: This is the preceding car in the target lane
    :return:success message
    """
    a = self.calculate_distance(car3.lat, car3.long, car2.lat, car2.long)
    if car1.lanechangestatus == 'complete':

        if car2.lanechangestatus == 'in progress':
            if line == 1:
                space = self.check_space(a)
```

Figure 5: Code Sniipet:Lane-Change Scenario [8]

4 Uppaal Model

In order to implement the verification of the system, an Uppaal Model was also created. We have been working on the concrete simulator as well but we were not not able to finalize it before our presentation. Nevertheless, figure 6 is the model view of the scenario taken as a screenshot but the source is inserted in the GitHub repository.

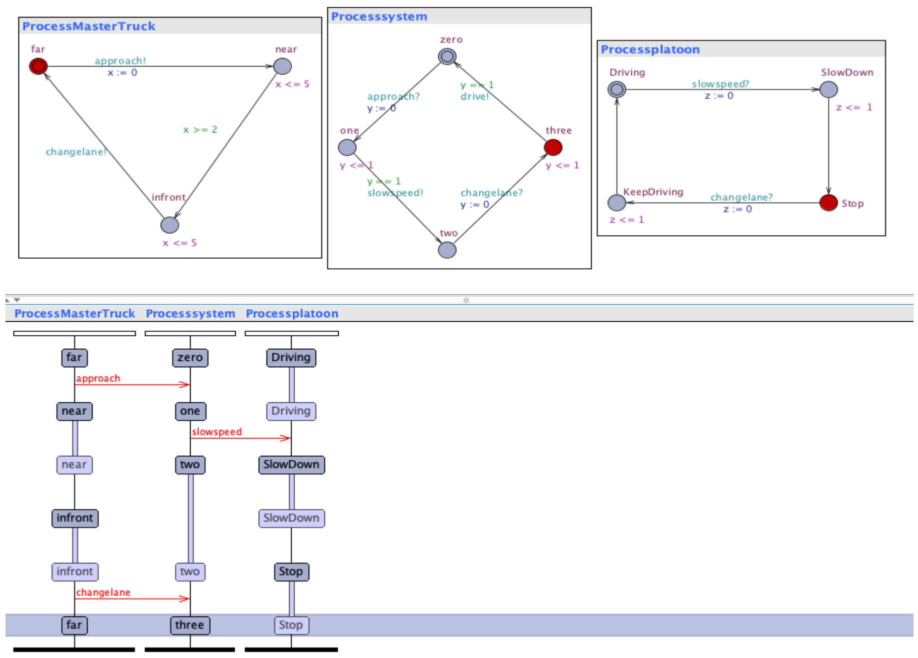


Figure 6: Uppaal Model

5 Leader Election

For our system's leader selection, we studied two algorithms: the Bully algorithm and the KNN algorithm. The reason behind approaching these two algorithm was to determine the master truck for the system. After the evaluation and results we finally select KNN algorithm for the selection of master truck in truck platooning system. Complete explanation of logic and code will be described in details later in the report.

5.1 Bully Algorithm

The algorithm is easy to understand if we look at the figure 3 so an election begins when the leader resigns or fails to respond. In the figure, we can see that node 7 steps down from his position, and Node 4 notices that the current leader is not responding and tries to become a leader itself, sending an Election message to all nodes with a higher value or level, for example, it will send message to node 5, node 6, and node 7 to see if they are working. Node 4 steps down from election when one of the larger nodes responds to node 4 about their presence. Nodes 5 and 6 now respond to Node 4 about their presence and begin competing in the election. Node 5 sends an election message to nodes 6 and 7, which checks their availability. Node 6 then responds to node 5 about its presence. Node 5 stops participating in this election after receiving a response from node 6. Node 6 broadcasts an election message to node 7 and awaits its response. The request is timed out after a certain amount of time, and node 6 realises that node 7 is down or not responding. Node 6 sends a Coordination message to all nodes because it has the highest rank (level) of all the available working nodes.

- Code Snippet for Election using Bully Algorithm [8] [5]

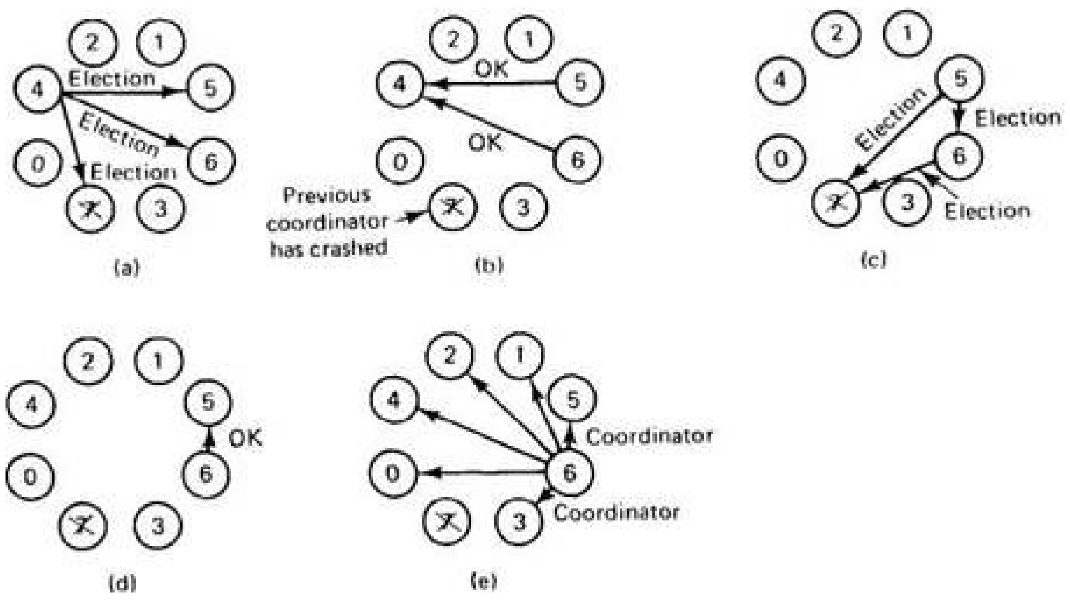
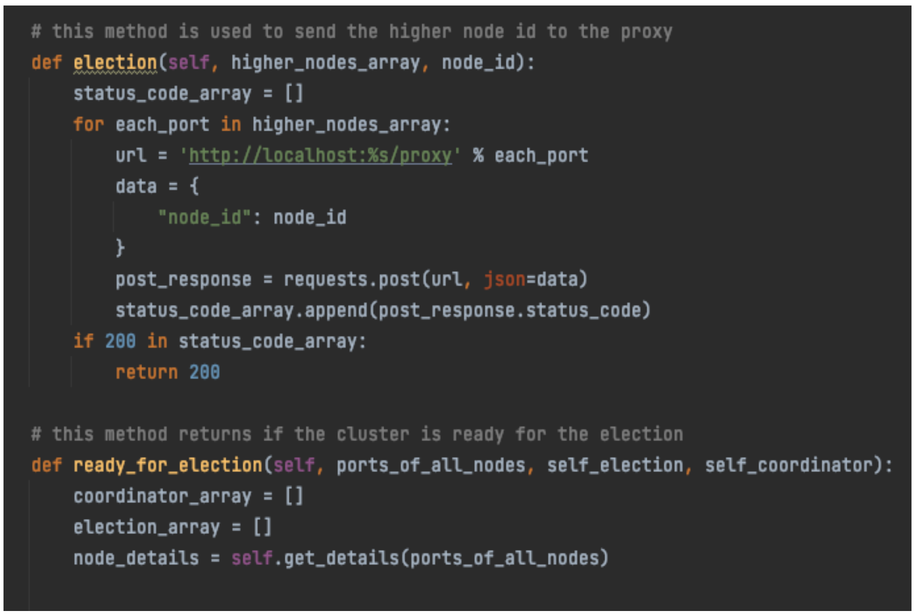


Figure 7: Bully Algorthim [7]



```
# this method is used to send the higher node id to the proxy
def election(self, higher_nodes_array, node_id):
    status_code_array = []
    for each_port in higher_nodes_array:
        url = 'http://localhost:%s/proxy' % each_port
        data = {
            "node_id": node_id
        }
        post_response = requests.post(url, json=data)
        status_code_array.append(post_response.status_code)
    if 200 in status_code_array:
        return 200

# this method returns if the cluster is ready for the election
def ready_for_election(self, ports_of_all_nodes, self_election, self_coordinator):
    coordinator_array = []
    election_array = []
    node_details = self.get_details(ports_of_all_nodes)
```

Figure 8: Uppaal Model [8]

5.2 KNN Algorithm

Before starting with the explanation that how we have utilized the KNN algorithm for our work , we will explain briefly what basically KNN is , what are the KNN parameters and what are the steps involved in the KNN algorithm. K-nearest neighbors (KNN) algorithm is a type of supervised machine learning algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. Lazy learning and non parametric learning are two main attractive properties of this algorithm that encourage the users to use this algorithm in machine learning. It does not have a specialized training phase and uses all the data for training while classification that is way we called it lazy learning algorithm. The behaviour of non-considering underlying data made it non parametric algorithm.

5.3 Basic Principle of KNN Algorithm

With the help of example, I would like to explain the logic and functionality of KNN. Think about the image in Figure 4 below. Let's imagine that we have drawn a two-dimensional feature space from the data points in our training set. We have a total of six data points, three of which are red and three of which are blue, as illustrated. The group1 data points are in red, and the group2 data points are in blue. The new point for which a class is to be predicted is represented by a yellow data point in a feature space. Naturally, since its closest neighbors are also red points, we claim that it belongs to group1.

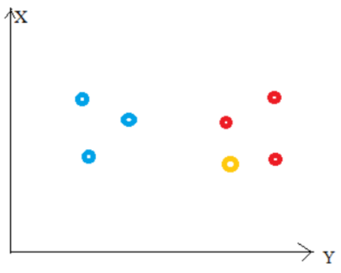


Figure 9: Feature Space for KNN [1]

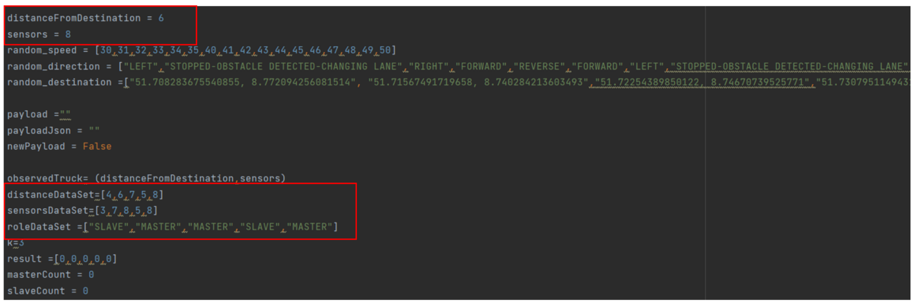
Nearest neighbors in this context are those data points that are the closest to our new data point in feature space and this is the main premise of this algorithm. K is the number of such data points that we take into account when we use the algorithm. Therefore, while utilizing the KNN method, the distance metric and K value are two crucial factors. The most often used distance metric is euclidean distance.

6 Coding Operations in our Truck Platooning Project

In this section we will explain the whole working and the coding approach of our work .The language that we are using for the programming of this code was python and for the communication among the client and host we were using the MQTT protocol. We have made 5 python scripts such as the Role assignment script, truck1 , truck 2, truck 3, Truck 4. In our logic we assign the truck there role during the run time. Paho-mqtt library we are using for the sake of establishing the mqtt connection and exchanging the information /data among the client and broker. Each of the truck contain its very own topic. To send or receive the information we are using the JSON payloads, in which we are saving the Truck ID , role, timestamp, distance of the truck and all these information. Each of the topic contain its own client and call backs function which are responsible for making the connection and gathering the data from the subscribed topic on which the payload is publishing the data. We are using the KNN algorithm for the selection of the master truck during the run time.

6.1 Selection of Master Truck using KNN Algorithm

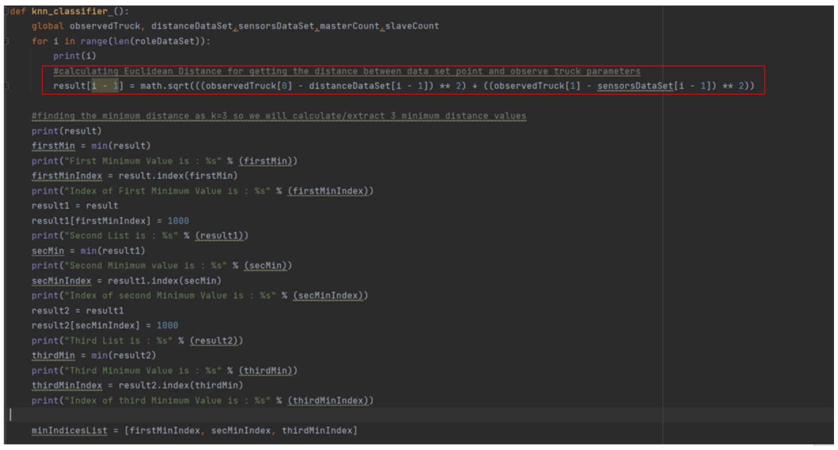
For electing the leader, we have set the logic that the leader truck will be the one with large number of sensor and have greater number of the distance from the destination, so in our case we have set the number of sensor and distance of the truck from the destination as we can see in the Figure.10 below . Also for electing the leader truck we will check the number of master counts and the truck which has the maximum number of master counts will get the role of Master /leading truck



```
distanceFromDestination = 6
sensors = 8
random_speed = [30,31,32,33,34,35,40,41,42,43,44,45,46,47,48,49,50]
random_direction = ["LEFT","STOPPED-OBSTACLE DETECTED-CHANGING LANE","RIGHT","FORWARD","REVERSE","FORWARD","LEFT","STOPPED-OBSTACLE DETECTED-CHANGING LANE"
random_destination =["51.708283675540855, 8.772094256081514", "51.71567491719658, 8.740284213603493","51.72254389850122, 8.74670739525771","51.730795114943

payload =""
payloadJson = ""
newPayload = False

observedTruck= (distanceFromDestination,sensors)
distanceDataSet=[4,6,7,5,8]
sensorsDataSet=[3,7,8,5,8]
roleDataSet =["SLAVE","MASTER","MASTER","SLAVE","MASTER"]
k=3
result =[0,0,0,0,0]
masterCount = 0
slaveCount = 0
```

Figure 10: Parameters for Master Election

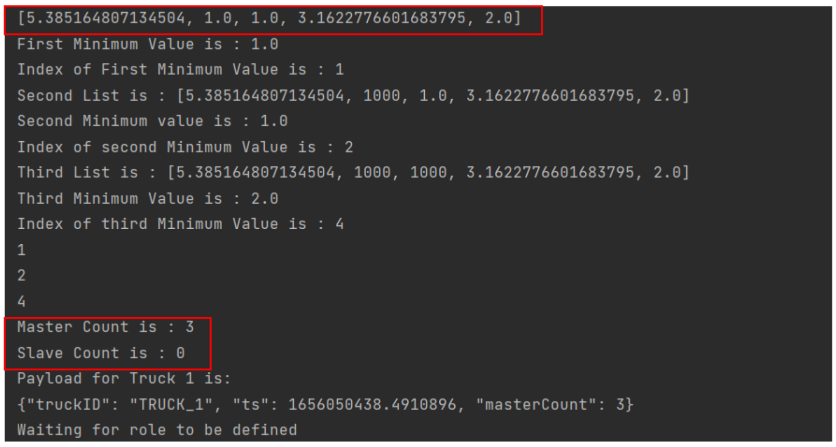
This is for the truck 1 and the same logic is used for the all other trucks as well with different number of sensor. I have highlighted the code in which we are giving the random value of sensor and distance from the destination and in he below highlighted area we have 3 datasets such as distance dataset , sensor dataset and the role dataset and on the base of these datasets we are going to compare the above set value of distance and sensors and value of K we set to 3 , the reason for setting the value as 3 is because our data set is small and contain only 5 values for each of the dataset so with the K=3 we get the best possible outcome for figuring or selecting the leading truck. Once doing all this we move to the main logic of the this whole project and the function we gave the name of KNN classifier.code snippet for these data centres are shown in Figure.6 below



```
def knn_classifier_():
    global observedTruck, distanceDataSet,sensorsDataSet,masterCount,slaveCount
    for i in range(len(roleDataSet)):
        print(i)
        #calculating Euclidean Distance for getting the distance between data set point and observe truck parameters
        result[i - 1] = math.sqrt(((observedTruck[0] - distanceDataSet[i - 1]) ** 2) + ((observedTruck[1] - sensorsDataSet[i - 1]) ** 2))

    #finding the minimum distance as k=3 so we will calculate/extract 3 minimum distance values
    print(result)
    firstMin = min(result)
    print("First Minimum Value is : %s" % (firstMin))
    firstMinIndex = result.index(firstMin)
    print("Index of First Minimum Value is : %s" % (firstMinIndex))
    result1 = result
    result1[firstMinIndex] = 1000
    print("Second List is : %s" % (result1))
    secMin = min(result1)
    print("Second Minimum value is : %s" % (secMin))
    secMinIndex = result1.index(secMin)
    print("Index of second Minimum Value is : %s" % (secMinIndex))
    result2 = result1
    result2[secMinIndex] = 1000
    print("Third List is : %s" % (result2))
    thirdMin = min(result2)
    print("Third Minimum Value is : %s" % (thirdMin))
    thirdMinIndex = result2.index(thirdMin)
    print("Index of third Minimum Value is : %s" % (thirdMinIndex))

    minIndicesList = [firstMinIndex, secMinIndex, thirdMinIndex]
```

Figure 11: Code Snippet for Data Centres

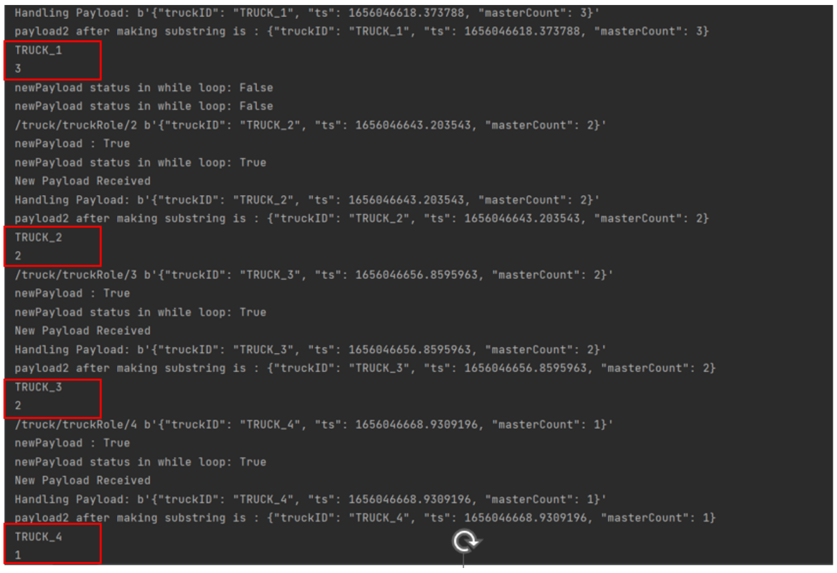
In this function what we are doing is first of all we find the encludien distance for finding the nearst neighboring points, this will give us 5 minimum value which are close to the define data set. Among these 5 value we are going to find the 3 least minimum values and check on which index we are getting the minimum value and once we get the minimum value we replace that value with the big number in our case with 1000 so in the next iteration we can get the next minimum value . Once all three minimum value been fetched out of the array we check the indexes and compare with role dataset to see how many time do we have get the master as a role and salve . This process will be done for all the remaining trucks and the truck with the most master count will become the leading truck , for the better understanding I have added the result screenshot's in Figure.12 below which can easily figure out the logic that has been discussed above



```
[5.385164807134504, 1.0, 1.0, 3.1622776601683795, 2.0]
First Minimum Value is : 1.0
Index of First Minimum Value is : 1
Second List is : [5.385164807134504, 1000, 1.0, 3.1622776601683795, 2.0]
Second Minimum value is : 1.0
Index of second Minimum Value is : 2
Third List is : [5.385164807134504, 1000, 1000, 3.1622776601683795, 2.0]
Third Minimum Value is : 2.0
Index of third Minimum Value is : 4
1
2
4
Master Count is : 3
Slave Count is : 0
Payload for Truck 1 is:
{"truckID": "TRUCK_1", "ts": 1656050438.4910896, "masterCount": 3}
Waiting for role to be defined
```

Figure 12: Result Screenshot for values of Distance and Master Counts

This is the result of truck 1 in which we have got the 5 minimum value in the array after the comparsion and then out of 5 we have taken the only 3 smallest value and print out there indexes and in this case the truck 1 has 0 salve count and 3 master count. The role of the truck is still in pending as first we have to run all the scripts and once for each script the master and the slave count been done then only the roles of the trucks been assigned. Below I have added one more screenshot of the truck in Figure.13 with the number of master and slave counts

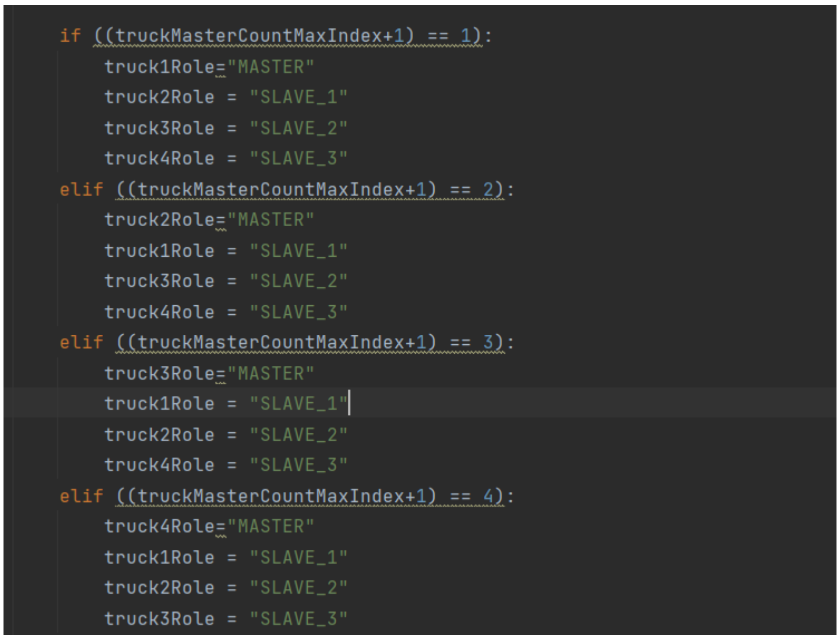


```
Handling Payload: b'{"truckID": "TRUCK_1", "ts": 1656046618.373788, "masterCount": 3}'
payload2 after making substring is : {"truckID": "TRUCK_1", "ts": 1656046618.373788, "masterCount": 3}
TRUCK_1
3
newPayload status in while loop: False
newPayload status in while loop: False
/truck/truckRole/2 b'{"truckID": "TRUCK_2", "ts": 1656046643.203543, "masterCount": 2}'
newPayload : True
newPayload status in while loop: True
New Payload Received
Handling Payload: b'{"truckID": "TRUCK_2", "ts": 1656046643.203543, "masterCount": 2}'
payload2 after making substring is : {"truckID": "TRUCK_2", "ts": 1656046643.203543, "masterCount": 2}
TRUCK_2
2
/truck/truckRole/3 b'{"truckID": "TRUCK_3", "ts": 1656046656.8595963, "masterCount": 2}'
newPayload : True
newPayload status in while loop: True
New Payload Received
Handling Payload: b'{"truckID": "TRUCK_3", "ts": 1656046656.8595963, "masterCount": 2}'
payload2 after making substring is : {"truckID": "TRUCK_3", "ts": 1656046656.8595963, "masterCount": 2}
TRUCK_3
2
/truck/truckRole/4 b'{"truckID": "TRUCK_4", "ts": 1656046668.9309196, "masterCount": 1}'
newPayload : True
newPayload status in while loop: True
New Payload Received
Handling Payload: b'{"truckID": "TRUCK_4", "ts": 1656046668.9309196, "masterCount": 1}'
payload2 after making substring is : {"truckID": "TRUCK_4", "ts": 1656046668.9309196, "masterCount": 1}
TRUCK_4
1
```

Figure 13: Master Counts for All trucks

6.2 Logic for Role Assigning in Truck Platooning

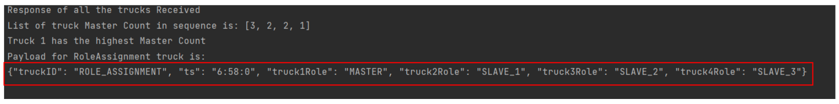
Once the count of master and slave trucks has been done then this data has been send to the "Role-assignment-Script " for defining the role of each and every truck. Below I have added the screenshot of the code in Figure.14 for the better understanding of the reader



```
if ((truckMasterCountMaxIndex+1) == 1):
    truck1Role="MASTER"
    truck2Role = "SLAVE_1"
    truck3Role = "SLAVE_2"
    truck4Role = "SLAVE_3"
elif ((truckMasterCountMaxIndex+1) == 2):
    truck2Role="MASTER"
    truck1Role = "SLAVE_1"
    truck3Role = "SLAVE_2"
    truck4Role = "SLAVE_3"
elif ((truckMasterCountMaxIndex+1) == 3):
    truck3Role="MASTER"
    truck1Role = "SLAVE_1"
    truck2Role = "SLAVE_2"
    truck4Role = "SLAVE_3"
elif ((truckMasterCountMaxIndex+1) == 4):
    truck4Role="MASTER"
    truck1Role = "SLAVE_1"
    truck2Role = "SLAVE_2"
    truck3Role = "SLAVE_3"
```

Figure 14: Code Snippet for Role Assigning

Here we are checking the indexes of the master count array , the index which contain the maximum number of the master count will become the leading truck and the rest of them will be the slave and keep following him until unless some other become the master but this is possible only when we stop the scripts and run them again because once the role has been assigned it will remain the same till the programm is running and not get terminated . This was also the one of the limitation of our system that during the run time there is no possibility of changing the master truck. So in our case the truck one contain the maximum number of master counts so it will become the leader truck and all the other will following the truck and as I have shown in the Figure.15 below the truck 1 is on index 0 with the value 3 so in the above (Figure.14) the first condition get true and Truck 1 will be assigned with the role of Leading or master truck

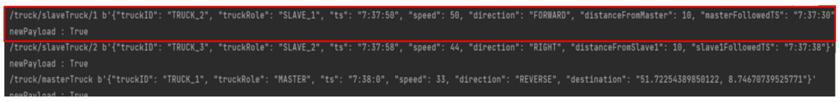


```
Response of all the trucks Received
List of truck Master Count in sequence is: [3, 2, 2, 1]
Truck 1 has the highest Master Count
Payload for RoleAssignment truck is:
{"truckID": "ROLE_ASSIGNMENT", "ts": "6:58:0", "truck1Role": "MASTER", "truck2Role": "SLAVE_1", "truck3Role": "SLAVE_2", "truck4Role": "SLAVE_3"}
```

Figure 15: Screenshorts for result of Role Assigning

6.3 Connectivity and Data Sharing between Following Trucks

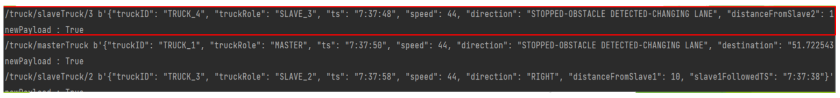
As we know that our first truck or you can say that truck1 with high number of master counts will be selected as master truck and it will contain the data and information of all following trucks. All the following trucks will follow the instructions of master trucks. Now we will see how following trucks communicate and check that either they will have data of all the trucks in the platoon or not? You can see the in the Figure.16 for the slave truck1 (or truck2), It have data of only truck which is behind it and in front of it. It have information and data of only two trucks which are master truck and slave 2. I have highlighted in figure below for the slave1.



```
/truck/slaveTruck/1 b'{"truckID": "TRUCK_2", "truckRole": "SLAVE_1", "ts": "7:37:50", "speed": 50, "direction": "FORWARD", "distanceFromMaster": 10, "masterFollowedTS": "7:37:30"
newPayload : True
/truck/slaveTruck/2 b'{"truckID": "TRUCK_3", "truckRole": "SLAVE_2", "ts": "7:37:58", "speed": 44, "direction": "RIGHT", "distanceFromSlave1": 10, "slave1FollowedTS": "7:37:38"}'
newPayload : True
/truck/masterTruck b'{"truckID": "TRUCK_1", "truckRole": "MASTER", "ts": "7:38:0", "speed": 33, "direction": "REVERSE", "destination": "51.72254389850122, 8.74670739525771"}'
```

Figure 16: Terminal screen short for Slave1/Truck2

Likewise the same pattern all the following trucks will have information of only two trucks which are in its front and back. For the last truck (truck4)in the platoon, It have the data of truck3 (or slave2) and there is no truck behind because it is the last truck of platoon so it will communicate and share data with master truck (truck1). You can see in the Figure.17 below.

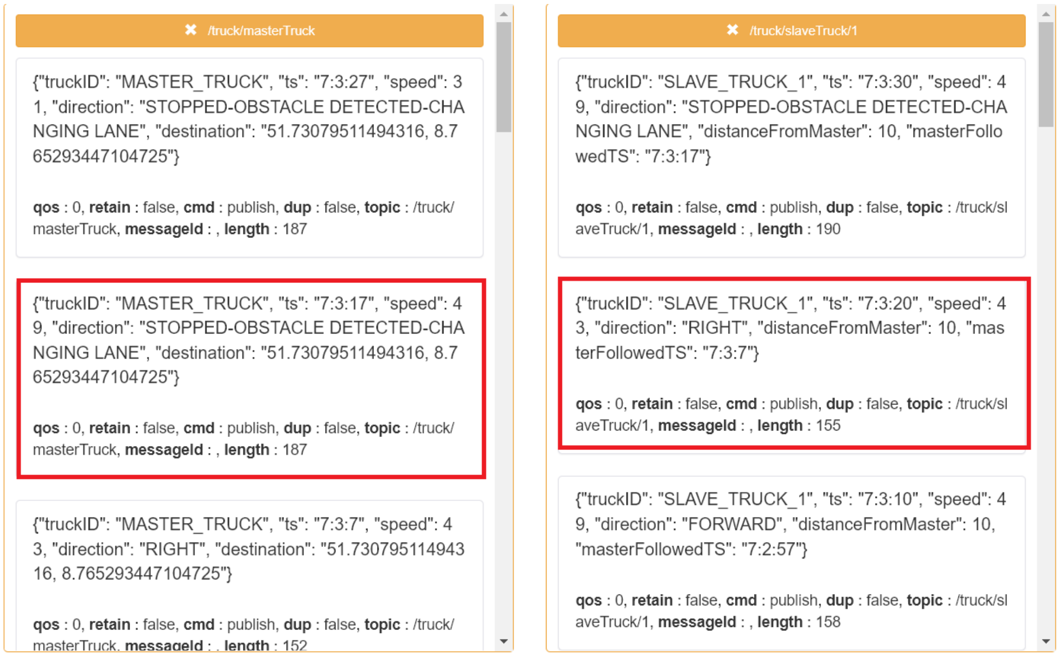


```
/truck/slaveTruck/3 b'{"truckID": "TRUCK_4", "truckRole": "SLAVE_3", "ts": "7:37:48", "speed": 44, "direction": "STOPPED-OBSTACLE DETECTED-CHANGING LANE", "distanceFromSlave2": 1
newPayload : True
/truck/masterTruck b'{"truckID": "TRUCK_1", "truckRole": "MASTER", "ts": "7:37:50", "speed": 44, "direction": "STOPPED-OBSTACLE DETECTED-CHANGING LANE", "destination": "51.722543
newPayload : True
/truck/slaveTruck/2 b'{"truckID": "TRUCK_3", "truckRole": "SLAVE_2", "ts": "7:37:58", "speed": 44, "direction": "RIGHT", "distanceFromSlave1": 10, "slave1FollowedTS": "7:37:38"}'
```

Figure 17: Terminal screen short for Slave3/Truck4

7 MQTT BOX RESULTS

MQTT box is a basically we are using MQTT protocol and this is the plugin, by the help of this the data we publish are displaying on the MQTT box , as we are working on the truck platooning the concept of truck platoon is it has one leading truck and every other follows the commands of leading truck, every truck has some parameters as we can see in figure in the red box the leading truck has ID, time spent , speed and destination but the destination will be only with the leading truck. Now the leading truck as shown in figure 4 has time 7:3:17 speed is 43 and direction is right ,now if we see in the red box 2, slave truck has time 7:3:20 with same parameters speed 43 direction is right and the distance from master is 10 meters but in end we can see it showing that the truck one is following time spent of master truck Now, if we look at



/truck/masterTruck

{"truckID": "MASTER_TRUCK", "ts": "7:3:27", "speed": 31, "direction": "STOPPED-OBSTACLE DETECTED-CHANGING LANE", "destination": "51.73079511494316, 8.765293447104725"}

qos : 0, **retain** : false, **cmd** : publish, **dup** : false, **topic** : /truck/masterTruck, **messageId** : , **length** : 187

{"truckID": "MASTER_TRUCK", "ts": "7:3:17", "speed": 49, "direction": "STOPPED-OBSTACLE DETECTED-CHANGING LANE", "destination": "51.73079511494316, 8.765293447104725"}

qos : 0, **retain** : false, **cmd** : publish, **dup** : false, **topic** : /truck/masterTruck, **messageId** : , **length** : 187

{"truckID": "MASTER_TRUCK", "ts": "7:3:7", "speed": 43, "direction": "RIGHT", "destination": "51.730795114943 16, 8.765293447104725"}

qos : 0, **retain** : false, **cmd** : publish, **dup** : false, **topic** : /truck/masterTruck, **messageId** : , **length** : 152

/truck/slaveTruck/1

{"truckID": "SLAVE_TRUCK_1", "ts": "7:3:30", "speed": 49, "direction": "STOPPED-OBSTACLE DETECTED-CHANGING LANE", "distanceFromMaster": 10, "masterFollowedTS": "7:3:17"}

qos : 0, **retain** : false, **cmd** : publish, **dup** : false, **topic** : /truck/slaveTruck/1, **messageId** : , **length** : 190

{"truckID": "SLAVE_TRUCK_1", "ts": "7:3:20", "speed": 43, "direction": "RIGHT", "distanceFromMaster": 10, "masterFollowedTS": "7:3:7"}

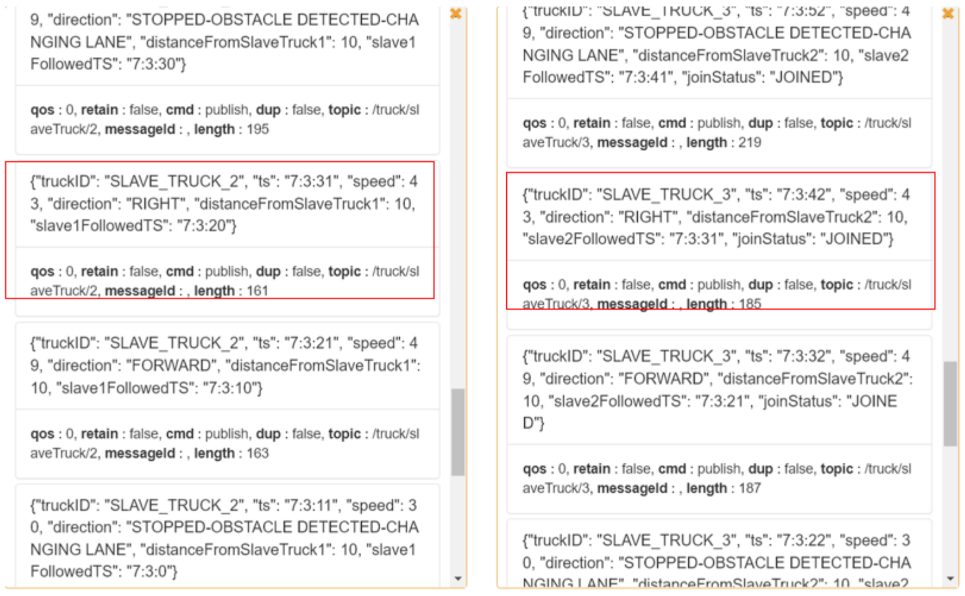
qos : 0, **retain** : false, **cmd** : publish, **dup** : false, **topic** : /truck/slaveTruck/1, **messageId** : , **length** : 155

{"truckID": "SLAVE_TRUCK_1", "ts": "7:3:10", "speed": 49, "direction": "FORWARD", "distanceFromMaster": 10, "masterFollowedTS": "7:2:57"}

qos : 0, **retain** : false, **cmd** : publish, **dup** : false, **topic** : /truck/slaveTruck/1, **messageId** : , **length** : 158

Figure 18: MQTT Box Result 1

figure 18 in the red box, we can see that truck 2 is following the time spent by slave 1, which is 7:3:20, and all other parameters are the same. With the same concept, the last truck will follow the time spent by truck 2, with all the same parameters, the speed is 43, and the direction is right. Another example in figure 6 will help us understand it better: if the master truck detects

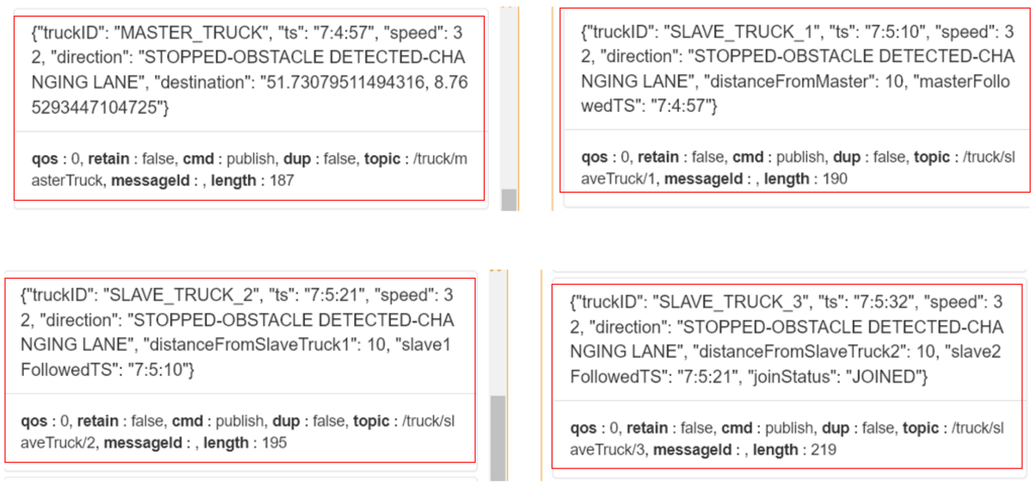


```
9, "direction": "STOPPED-OBSTACLE DETECTED-CHA
NGING LANE", "distanceFromSlaveTruck1": 10, "slave1
FollowedTS": "7:3:30"}

qos : 0, retain : false, cmd : publish, dup : false, topic : /truck/sl
aveTruck/2, messageId : , length : 195

{"truckID": "SLAVE_TRUCK_2", "ts": "7:3:31", "speed": 4
3, "direction": "RIGHT", "distanceFromSlaveTruck1": 10,
"slave1FollowedTS": "7:3:20"}

qos : 0, retain : false, cmd : publish, dup : false, topic : /truck/sl
aveTruck/2, messageId : , length : 161

{"truckID": "SLAVE_TRUCK_2", "ts": "7:3:21", "speed": 4
9, "direction": "FORWARD", "distanceFromSlaveTruck1":
10, "slave1FollowedTS": "7:3:10"}

qos : 0, retain : false, cmd : publish, dup : false, topic : /truck/sl
aveTruck/2, messageId : , length : 163

{"truckID": "SLAVE_TRUCK_2", "ts": "7:3:11", "speed": 3
0, "direction": "STOPPED-OBSTACLE DETECTED-CHA
NGING LANE", "distanceFromSlaveTruck1": 10, "slave1
FollowedTS": "7:3:0"}
```

```
{"truckID": "SLAVE_TRUCK_3", "ts": "7:3:52", "speed": 4
9, "direction": "STOPPED-OBSTACLE DETECTED-CHA
NGING LANE", "distanceFromSlaveTruck2": 10, "slave2
FollowedTS": "7:3:41", "joinStatus": "JOINED"}

qos : 0, retain : false, cmd : publish, dup : false, topic : /truck/sl
aveTruck/3, messageId : , length : 219

{"truckID": "SLAVE_TRUCK_3", "ts": "7:3:42", "speed": 4
3, "direction": "RIGHT", "distanceFromSlaveTruck2": 10,
"slave2FollowedTS": "7:3:31", "joinStatus": "JOINED"}

qos : 0, retain : false, cmd : publish, dup : false, topic : /truck/sl
aveTruck/3, messageId : , length : 185

{"truckID": "SLAVE_TRUCK_3", "ts": "7:3:32", "speed": 4
9, "direction": "FORWARD", "distanceFromSlaveTruck2":
10, "slave2FollowedTS": "7:3:21", "joinStatus": "JOINE
D"}

qos : 0, retain : false, cmd : publish, dup : false, topic : /truck/sl
aveTruck/3, messageId : , length : 187

{"truckID": "SLAVE_TRUCK_3", "ts": "7:3:22", "speed": 3
0, "direction": "STOPPED-OBSTACLE DETECTED-CHA
NGING LANE", "distanceFromSlaveTruck2": 10, "slave2
```

Figure 19: MQTT Box Result 2

a lane change or an obstacle, the truck one will execute the same command with all parameters. As we can see, the truck has changed lanes, and trucks 2 and 3 are following it.



```
{"truckID": "MASTER_TRUCK", "ts": "7:4:57", "speed": 3
2, "direction": "STOPPED-OBSTACLE DETECTED-CHA
NGING LANE", "destination": "51.73079511494316, 8.76
5293447104725"}

qos : 0, retain : false, cmd : publish, dup : false, topic : /truck/m
asterTruck, messageId : , length : 187
```

```
{"truckID": "SLAVE_TRUCK_1", "ts": "7:5:10", "speed": 3
2, "direction": "STOPPED-OBSTACLE DETECTED-CHA
NGING LANE", "distanceFromMaster": 10, "masterFollo
wedTS": "7:4:57"}

qos : 0, retain : false, cmd : publish, dup : false, topic : /truck/sl
aveTruck/1, messageId : , length : 190
```

```
{"truckID": "SLAVE_TRUCK_2", "ts": "7:5:21", "speed": 3
2, "direction": "STOPPED-OBSTACLE DETECTED-CHA
NGING LANE", "distanceFromSlaveTruck1": 10, "slave1
FollowedTS": "7:5:10"}

qos : 0, retain : false, cmd : publish, dup : false, topic : /truck/sl
aveTruck/2, messageId : , length : 195
```

```
{"truckID": "SLAVE_TRUCK_3", "ts": "7:5:32", "speed": 3
2, "direction": "STOPPED-OBSTACLE DETECTED-CHA
NGING LANE", "distanceFromSlaveTruck2": 10, "slave2
FollowedTS": "7:5:21", "joinStatus": "JOINED"}

qos : 0, retain : false, cmd : publish, dup : false, topic : /truck/sl
aveTruck/3, messageId : , length : 219
```

Figure 20: MQTT Box Result 3

8 Summary

In light of this paper, we have learnt the art of planning designing, validation, verification and implementation of a real world scenario. Moreover, in order to check the our model-based approach usage of Uppaal was performed. Deep hands-on with python to implement the scenario. Conclusively, learning new simulation methods for electing a leader and to portray the scenarios is a threshold step in the era of industry 4.0.

9 Appendix

DECLARATION

We, Zahid Pervaz, Muhammad Rohail Usman, Danish Zaheer Malik, and Qamar Sajjad of Hochschule Hamm Lippstadt, hereby declare that the project ”Truck Platooning” was successfully completed under the supervision of Prof. Dr. Martin Hirsch and Prof. Dr. Stefan Henkler, Department Lippstadt 2, Hochschule Hamm Lippstadt. This project report was created as part of the Autonomous Systems 1 Labwork. We, Zahid Pervaz, Muhammad Rohail Usman, Danish Zaheer Malik, and Qamar Sajjad, herewith declare that we have composed the present paper and work by myself and without the use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form have not been submitted to any examination body and have not been published. This paper was not yet, even in part, used in another examination or as a course performance. We agree that my work may be checked by a plagiarism checker.

[illegible]

Figure 21: Muhammad Rohail Usman 10.07.2022

Zahid Parvez

Figure 22: Zahid Parvez 10.07.2022

[illegible]

Figure 23: Qamar Sajjad 10.07.2022

[illegible]

Figure 24: Danish Zaheer Malik 10.07.2022

9.1 Contribution

As a four person group, we each put in an equal amount of time and effort on each component of the project. Every group member contributed to their best and equally, we used online tools Anydesk and Google Meet to focus on each milestones. We worked on the final simulation part in a collective manner, having inputs from each individual member and then devising the final result.

9.2 History of Git Repository

The GitHub repository commits reflect the input made by the group. Starting on with the scenario description on 07 April, 2022 up until the day of submission the inputs from individual members can be seen. In order to keep the task distribution aligned each member made their efforts shown via individual commits. GitHub folder of Lane-Change Scenario was an effort by Muhammad Rohail Usman for his scenario. Qamar Sajjad and Danish Zaheer Malik worked on initial diagrams that are made at early stages of the project to check requirements and conditions are uploaded on github. Zahid Parver and Rohail Usman worked on KNN Algorithmic approach for master election. Role assignment and Communication between the slave trucks are made by Qamar Sajjad and Danish Zaheer.

9.3 References

References

[1] patwardhan-2022, title=KNN algorithm: What is KNN algorithm: How does Knn function, url=https://www.analyticsvidhya.com/blog/2021/04/simple-understanding-and-implementation-of-knn-algorithm/, journal=Analytics Vidhya, author=Patwardhan, Sai, year=2022, month=Jun

[2] shi-018, title=Cooperative Platooning and Lane Changing for Connected and Automated Vehicles: An Entropy-Based Method, url=https://www.acsu.buffalo.edu/ qinghe/thesis/2018-08-01, journal=Thesis Yunpeng Shi - University at Buffalo, author=Shi, Yunpeng, year=2018, month=Jun

[3] renwu-021, title=Improving the Safe Operation of Platoon Lane Changing for Connected Automated Vehicles: A Novel Field-Driven Approach ,url=https://www.mdpi.com/2076-3417/11/16/7287, author=Wu,Renfei, Li, Linheng, Lu, Wenqi, Rui, Yikang and Ran, Bin, year=2021, month=Nov

[4] salsha-018, title=Evaluating Traffic Efficiency and Safety by Varying Truck Platoon Characteristics in a Critical Traffic Situation, url=https://www.researchgate.net/publication/339337014, author=Sharma,Salil, Snelder,Maaike, Klunder, Gerdien, and L.A.,Tavasszy, year=2020, month=Jul

[5] uyan-020, title=Electing master node in a cluster using Bully Algorithm url=https://isuruuy.medium.com/electing-master-node-in-a-cluster-using-bully-algorithm-b4e4fa30195c, author=Uyanage, Isuru, year=2020, month=Jun

[6] pat-21, title=Robust Python: Write Clean and Maintainable Code url=https://learning.oreilly.com/library/view/robust-python/9781098100650/preface01.html, author=Viafore, Patrick, year=2021, month=Jul

[7] Bully algorithm in Java - Javatpoint. (n.d.). Www.Javatpoint.Com. https://www.javatpoint.com/bully-algorithm-in-java

[8] autosys-2022, title=GitHub Repository: Autonomous Systems url=https://github.com/Rohail999/Autonomous_Systems_Lab_SS22/tree/main/Lane_Change_Scenario, year=2022, month=April