Autonomous Truck Platooning

Asm Nurussafa [1], Tasawar Siddiquy [2], Arfat Kamal [3]

Contents

1 Motivation [Asm Nurussafa] 3

2 Scenarios 4

2.1 Overall scenario 4

2.1.1 Requirements diagram [Asm Nurussafa] 4

2.1.2 Block Diagram [Tasawar Siddiquy] 5

2.2 Use Case Diagram [Arfat Kamal] 6

2.3 Platooning [Arfat Kamal] 7

2.4 Maintaining constant distance [Tasawar Siddiquy] 8

2.5 Intruder vehicle [Asm Nurussafa] 9

3 Timed Automata- UPPAAL 10

3.1 Platooning [Asm Nurussafa] 10

3.2 Maintaining constant distance [Tasawar Siddiquy] 12

4 Integrating Machine Learning Algorithm [Tasawar Siddiquy] 13

5 Implementation and Simulation [Asm Nurussafa] 14

5.1 Generating a map 15

[1] asm.nurussafa@stud.hshl.de
[2] tasawar.siddiquy@stud.hshl.de
[3] arfat.kamal@stud.hshl.de

5.2 Generating a router file 16
5.3 Configuration 16
5.4 Python script using TraCI 17
5.5 Implementing Platooning using Simpla 18

6 Conclusion **18**

7 Appendix **19**
7.1 Declaration of Originality [Asm Nurussafa] 19
7.2 Declaration of Originality [Tasawar Siddiquy] 20
7.3 Declaration of Originality [Arfat Kamal] 20
7.4 Task Responsibilities 20
7.5 History of GitHub upload 20

Abstract: One of the most important application areas in building a Smart City and a fully autonomous road-sytem is to implement autonomous truck platooning. In this paper, we will briefly discuss our motivation for this project. Additionally, we will discuss how we can model such a system using SysML diagrams and UPPAAL and how we can use machine learning algorithms in such scenarios. Finally, we integrate all of these ideas from our models and realize our system and simulate it in SUMO, a simulation tool.

Keywords: Autonomous Truck Platooning, SysML, UPPAAL, machine learning and SUMO.

1 Motivation [Asm Nurussafa]

For the movement of trucks on highways, truck platooning is a solution. Making trucks drive behind one another in a highway lane with a certain spacing between them is a relatively straightforward solution for maintaining safe clearance. The trucks coordinate with one another and move at the same speed. Vehicles exchange data and messages as part of this coordination. Significant advantages of this technique include decreased air drag, which improves fuel efficiency. Additionally, it makes better use of the resources provided by the roads, lessens traffic congestion, and reduces accidents.

Truck platoon systems are already available and in use at a number of automotive firms, including Volvo, Volkswagen, Scania, and others. The entire truck platooning system is a complicated one. It consists of numerous distinct systems, and each vehicle is a separate system. They create a truck platooning system collectively. But developing such intricate systems in systems comes with integration, security, and scalability issues.

In this paper, we will discuss how we can design such systems using SysML diagrams, timing constraints using timed automata in UPPAAL modelling system, discuss how machine learning can be integrated in such systems and finally, simulate the system in SUMO (Simulation of Urban mobility) using python scripts.

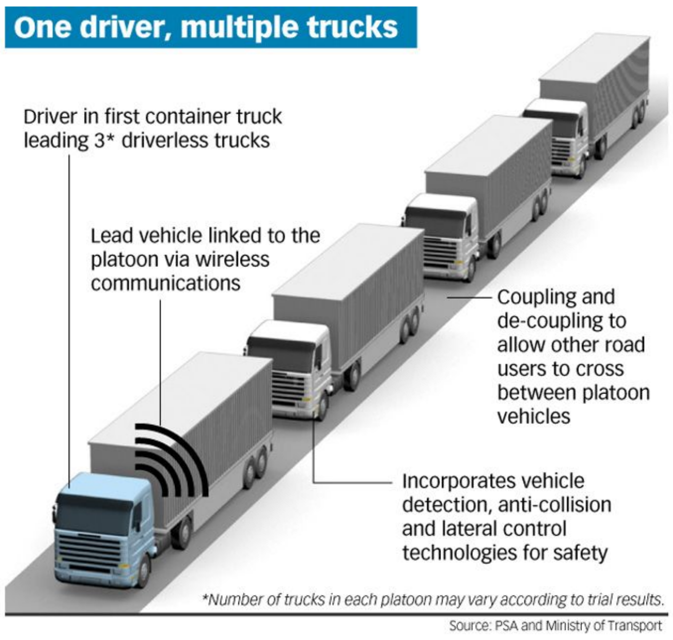

One driver, multiple trucks
Driver in first container truck leading 3* driverless trucks
Lead vehicle linked to the platoon via wireless communications
Coupling and de-coupling to allow other road users to cross between platoon vehicles
Incorporates vehicle detection, anti-collision and lateral control technologies for safety
*Number of trucks in each platoon may vary according to trial results.
Source: PSA and Ministry of Transport

Fig. 1: Autonomous Truck Platooning

2 Scenarios

2.1 Overall scenario

We used modeling diagrams to describe our scenarios.

2.1.1 Requirements diagram [Asm Nurussafa]

The first step in realizing our goals of this project, we considered an abstract level of requirements in SysML modelling. In system engineering, this is one of the most crucial steps. This would later help us in identifying and defining requirements for our goals in a precise manner. This diagram would help us later and validating and verifying our goals and later, concretize them altogether. For our requirements diagrams, we have created the requirements in a hierarchical manner. At the very top, we have the main requirement for our project, i.e. *Platooning*.

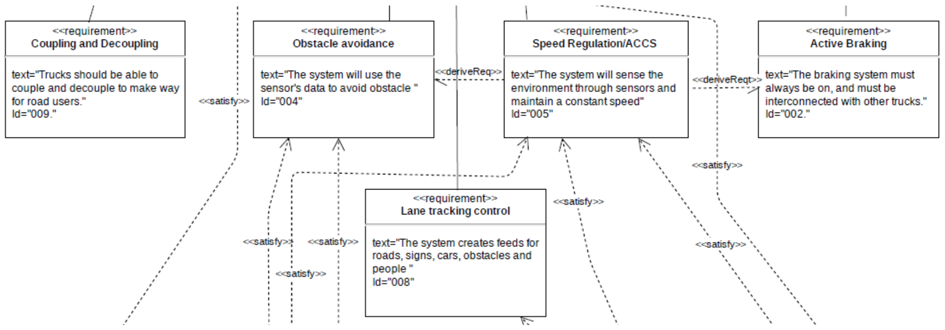

<<requirement>>
Coupling and Decoupling
text="Trucks should be able to couple and decouple to make way for road users."
Id="009."
<<requirement>>
Obstacle avoidance
text="The system will use the sensor's data to avoid obstacle "
Id="004"
<<requirement>>
Speed Regulation/ACCS
text="The system will sense the environment through sensors and maintain a constant speed"
Id="005"
<<requirement>>
Active Braking
text="The braking system must always be on, and must be interconnected with other trucks."
Id="002."
<<requirement>>
Lane tracking control
text="The system creates feeds for roads, signs, cars, obstacles and people "
Id="008"
<<deriveReq>>
<<deriveReqt>>
<<satisfy>>
<<satisfy>>
<<satisfy>>
<<satisfy>>
<<satisfy>>
<<satisfy>>
<<satisfy>>

Fig. 2: Requirements diagram.

Next, as seen in the following fig. 2, we have the requirements for *Coupling and Decoupling*, that should allow trucks to couple and decouple according to need. We, then, have *obstacle avoidance*, where the trucks in a platoon should be able to avoid any interference. The *speed regulation or ACCS* (Adaptive Cruise Control System), that should sense the environment with the help of sensors and maintain a constant speed. *Active braking* would help all the trucks in a platoon to brake in a synchronous manner. Lastly, in this layer we have the *lane-tracking control*. For the next level in hierarchy, as shown in 3, where we considered the hardware level requirements. For realizing this, we considered having a *GPS* (Global Positioning System), that would help the trucks to navigate by synchronizing location, velocity and time data. *Radar and Ultrasonic* sensors would help us in measuring the distance, angle and radial velocity using radio and ultrasonic waves. Lastly, *cameras* and as a medium of communication , *V2V* (Vehicle to vehicle) and *WLAN* (Wireless LAN). A full view of the requirements diagram can be seen in our github repository [**?**]

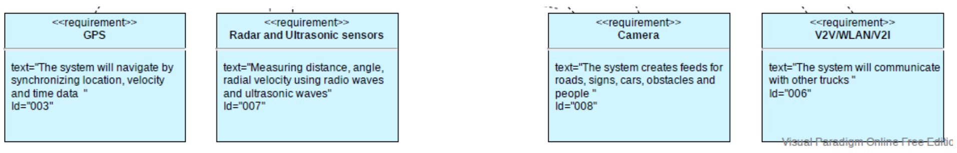

<<requirement>>
GPS
text="The system will navigate by synchronizing location, velocity and time data "
Id="003"
<<requirement>>
Radar and Ultrasonic sensors
text="Measuring distance, angle, radial velocity using radio waves and ultrasonic waves"
Id="007"
<<requirement>>
Camera
text="The system creates feeds for roads, signs, cars, obstacles and people "
Id="008"
<<requirement>>
V2V/WLAN/V2I
text="The system will communicate with other trucks "
Id="006"
Visual Paradigm Online Free Editio

Fig. 3: Next hierarchical layer in Requirements diagram.

2.1.2 Block Diagram [Tasawar Siddiquy]

A block Diagram provides a high-level overview of central system components, key process participants, and important functional relationships.

We can get a summary of our entire system in Figure 4. Our primary unit is our main system block or vehicle block. The system block will be linked to all other hardware and sensors. The block diagram also depicts the relationships between the blocks and the amount of hardware we need. Synchronization with the other cars will be provided through real-time cloud supervision, which will contain certain crucial traffic data and potential traffic alerts to avoid accidents. Local area connections between the vehicles can be made via WLAN. The effective autonomous braking system uses an active brake controller.

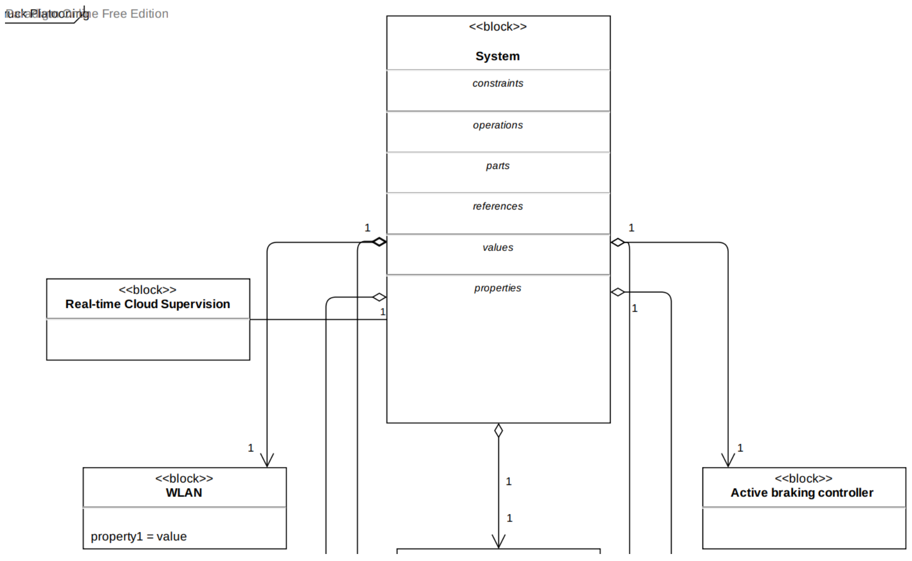


Fig. 4: Block Diagram Part 1

The major focus of Figure 5 was on crucial sensors for autonomous driving. A 3D virtual environment that has been mapped from the real world is especially created using lidar. A camera will enhance the capacity to sense other vehicles ahead of you while still using radar to detect them.

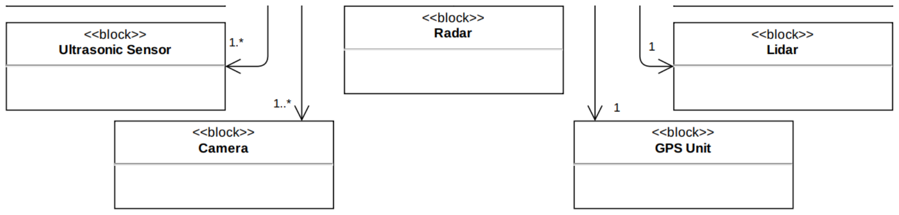


Fig. 5: Block Diagram Part 2

Ultrasonic sensors have been used to find things so that certain actions may be taken based on the data. The most important piece of equipment for truck platooning is a GPS unit. GPS is primarily used to find and communicate the location of the car to other vehicles.

2.2 Use Case Diagram [Arfat Kamal]

Use Case Diagram demonstrates the use cases of the system from a high level. It is a common practice to make an UML Use Case diagram at analysis phase of developing a system. Use case diagram gives a clear view of how other diagrams like activity diagram and sequence diagram should be created.

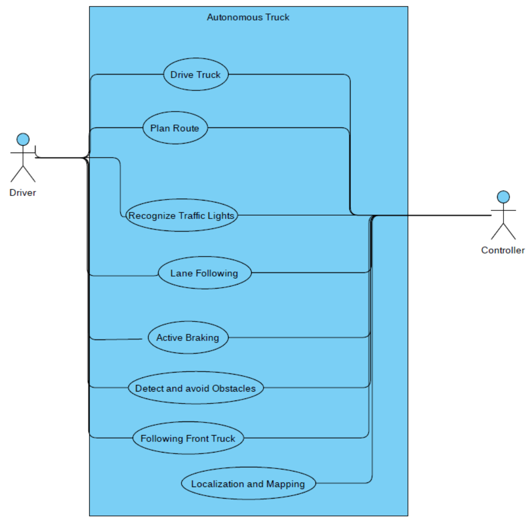


Fig. 6: Use Case Diagram

In Figure 6 an use case diagram can be found which was created with the goal of seeing the overall problem of our truck platooning project. In the diagram, there are two actors, driver and the controller. The controller is responsible for autonomously taking decisions in order to drive the truck autonomously. We considered both of these actors since we thought of it as a semi autonomous truck where driving manually and autonomously, both scenarios are possible. Most of the use cases are shared by both of the actors, such as driving the truck, planning route, recognizing traffic lights and so on.

2.3 Platooning [Arfat Kamal]

In truck platooning, coupling means when two trucks couple or pair with each other and the following truck(the truck which is behind) starts to follow the leading truck. Once the trucks are coupled, the driver of the following truck no longer needs to drive the truck, the truck would simply follow the the leading truck and drive autonomously until they are decoupled or disconnected.

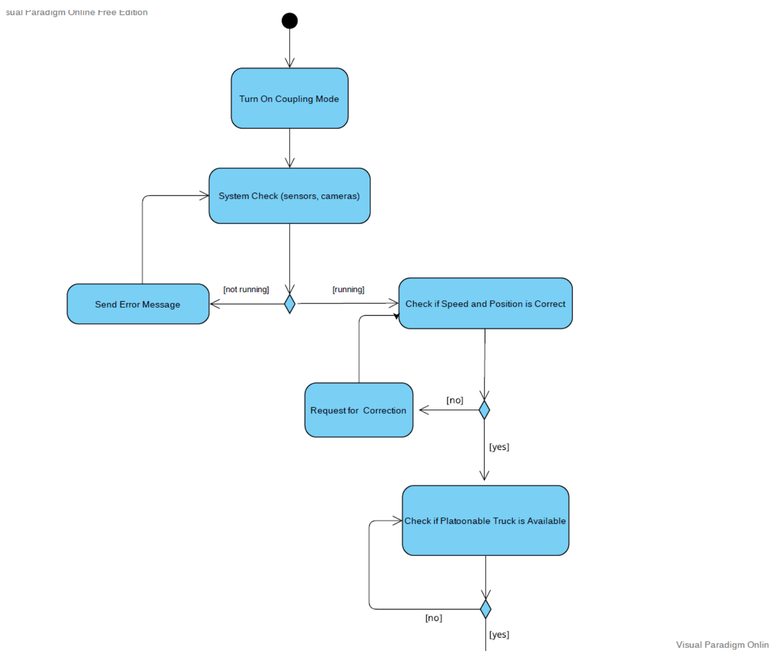

sual Paradigm Online Free Edition
Turn On Coupling Mode
System Check (sensors, cameras)
Send Error Message
[not running]
[running]
Check if Speed and Position is Correct
Request for Correction
[no]
[yes]
Check if Platoonable Truck is Available
[no]
[yes]
Visual Paradigm Onlin

Fig. 7: Coupling Activity Diagram

Figure 12 is a demonstration of how the coupling occurs using an activity diagram. This is a scenario when a following truck wants to find and couple with a leading truck. To do that, the driver first needs to turn on the coupling mode. Then the truck will check its system if everything is running properly, the sensors, cameras for example. If everything is running fine it will go to the next step which is checking if the speed and the position of the truck is suitable for platooning. If yes, it will look for a platoonable truck nearby. Once it finds one, it will send to the other truck a request to platoon. If the driver of the leading truck accepts that request, then the trucks can proceed for platooning.

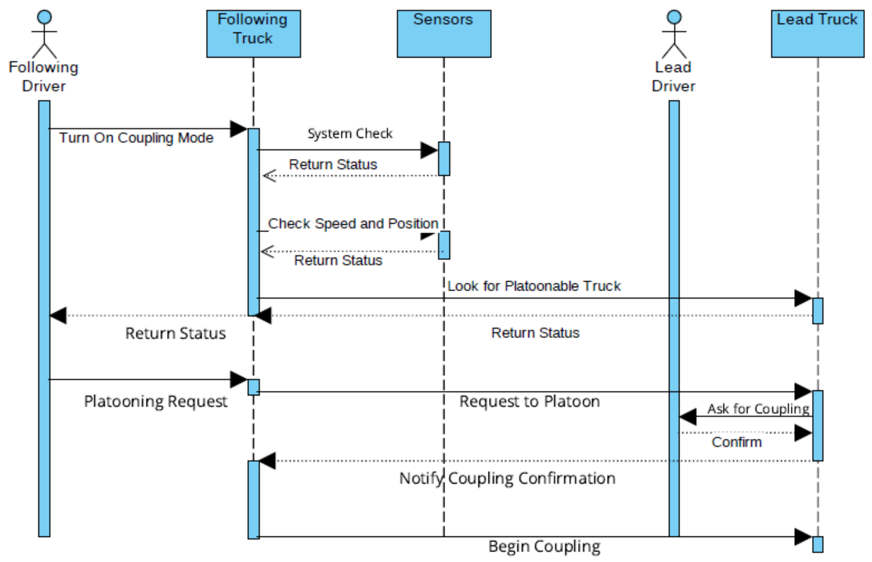


Fig. 8: Coupling Sequence Diagram

Figure 8 is depicts how one following truck can be coupled with a leading truck. It actually demonstrated the same steps as the activity diagram but more in details. The blue bar here represents how long an action is active so we can understand the timing behavior and sequencial order of the processes too.

2.4 Maintaining constant distance [Tasawar Siddiquy]

A sequence diagram shows how objects in a system interact with each other. It also represents the time sequence in a system and the scenarios of a system.

Figure 9 depicts the communication and timing behavior between trucks, sensors, and actuators. Additionally, the order of events has been noted. After entering the platoon, all of the vehicles will read the sensor data. The sensor data reading is displayed repeatedly so that this cycle can continue without interruption.

Figure 10 provided a detailed explanation of the communication process or the message passing between the vehicle and the actuators. Our system must adhere to a deadline when executing tasks otherwise the results may be harmful. For instance, we set each deadline to occur within 500 milliseconds in order to carry out a certain action, such as an automated

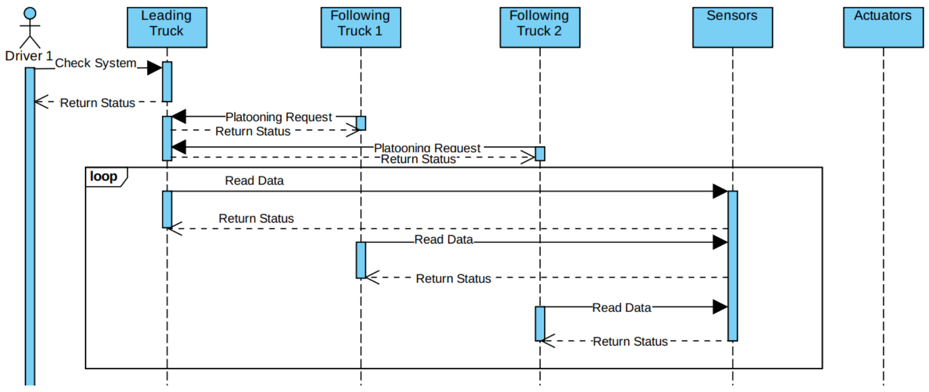


Fig. 9: Sequence Diagram Part 1

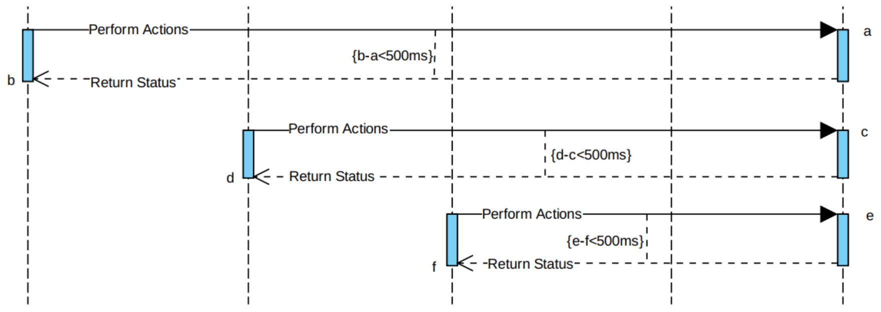


Fig. 10: Sequence Diagram Part 2

brake or emergency brake. The driver of the leading car will have the ability to manually override the system if required.

2.5 Intruder vehicle [Asm Nurussafa]

This was one of the initial scenarios we planned to implement, but later, had to leave it out during the implementation phase because of time constraints. In this scenario, we considered about the possibilities that might arise during platooning. As the name suggests, we consider an intruder vehicle that enters between two trucks which are in platooning mode. To explain and realize the scenario better, we created a SysML Sequence diagram, to help portray the series of actions and time constraints. In the following fig. 11, we consider the series of actions that occur when the platooning trucks detect the incoming of an intruder vehicle.

Here, the *sensors and actuators* depict all of the sensors and actuators of all the trucks in the platoon. So, initially, when any one of trucks detects an incoming of vehicle in between

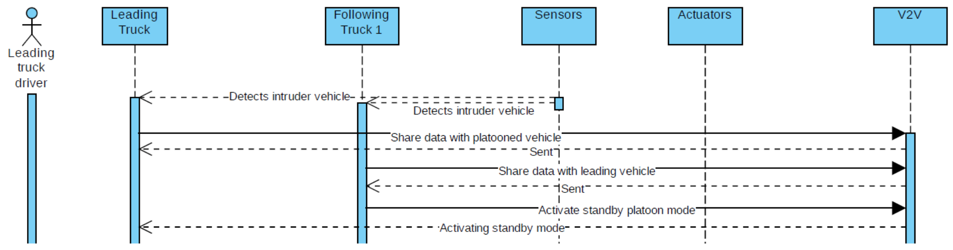


Fig. 11: Sequence diagram- Incoming vehicle.

them, they instantly share this information with other trucks, esp. the leading truck via V2V. Next, they all initialize a *standby platoon mode* as shown in the figure 11

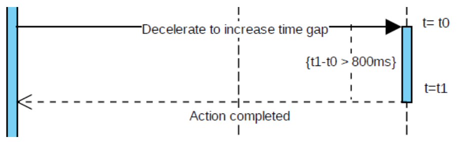


Fig. 12: Decelerating timing constraint.

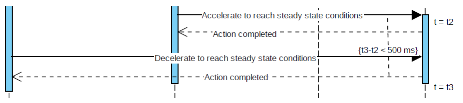

Accelerate to reach steady state conditions
t = t2
Action completed
{t3-t2 < 500 ms}
Decelerate to reach steady state conditions
Action completed
t = t3

Fig. 13: Accelerating timing constraint.

After that, in order to make space for the incoming vehicle, the trucks would have to decelerate and this shown in fig. 12. A timing constraint is also considered for the following series of action as shown. So, from the beginning of the deceleration, $t0$, and ending of this action and passing this information , $t1$, should take about 800 ms.
After that, the same series of actions take place when the trucks detect the outgoing of the vehicle and then, switch to *steady-state platooning mode*. The following trucks accelerate to gain back the space, as shown in fig. 13. The timing constraint we have considered for this action is about 500 ms.
A view of the full sequence diagram can be seen in our github repository [**?**]

3 Timed Automata- UPPAAL

Real-time systems are the major application for timed automata. To demonstrate the communication and control characteristics of our system, we employed timed automata.

3.1 Platooning [Asm Nurussafa]

Next, we dive into a bit of depth by modelling our *platooning* scenario using the UPPAAL modelling scheme. This would help us visualize and realize the real-time communications between different components. In this part, we will try and explain the diagram we have produced and it works. At the beginning, we have the behavior for our leading truck, as shown in fig. 14 The states, channels used for communications and guards are highlighted in parantheses here.

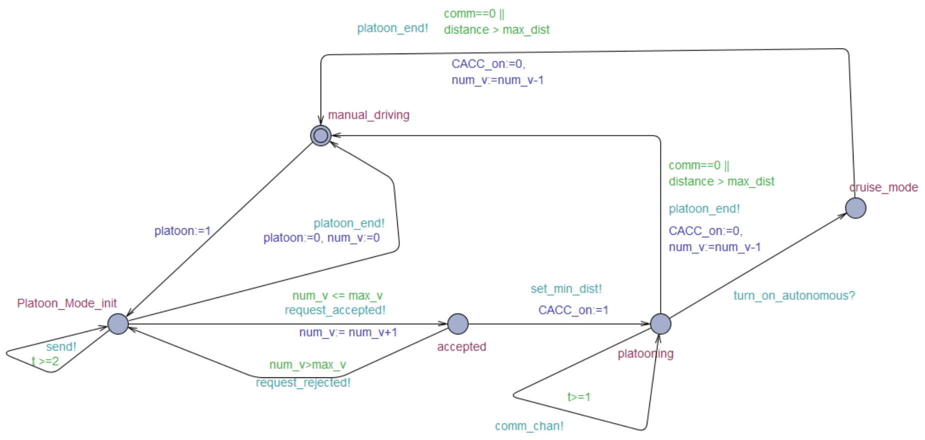


Fig. 14: UPPAAL model- Leading truck.

Initially, all of the trucks remain at *manual mode*. The leading truck then initializes *platooning mode* ($Platoon_Mode_init$) . Every two seconds, this is broadcasted to itself. If it deactivates ($platoon_end!$) this mode, the truck goes back to the manual mode. If it accepts ($request_accepted!$) it joins with other trucks and stays in *platoon* state. If not, it goes back to its former state. This transition also uses a guard ($num_v <= max_v$) , to check whether the maximum number of platooning car has been exceeded or not, as for instance, we have used the maximum number of platooning cars to be 7.
As an optional feature, we have added the case where the platoon can drive fully autonomous by transitioning to a *cruise mode* state. Additionally, if there is any communcation failure ($comm == 1$) during the platooning mode or the truck decides to stop platooning/ reaches destination, the truck goes back to *manual driving* state ($platoon_end!$).

Next, we take a look at the behavior of the following truck and communications with the leading truck as depicted in fig. 15

By observing, we can see that initially it is the same, i.e it starts at the *manual driving* state. It, then, enters the *decision node* state it wants to request for platooning to the leading truck, only if the guard ($platoon == 1$) is true. If the request is accepted by the leading truck, it enters the *platooning mode* state. If the request is rejected or it decides to end the platoon, it goes back to the *manual driving* state. Following this, it sets a minimum distance (set_min_dist) and activates adaptive cruise control ($CACC_on := 1$). If there is a communication failure ($comm_failure$) , it can choose to go to the *cruise mode* and from there, can decide to go back to the *manual driving* state using the reset ($reset$) communication channel.

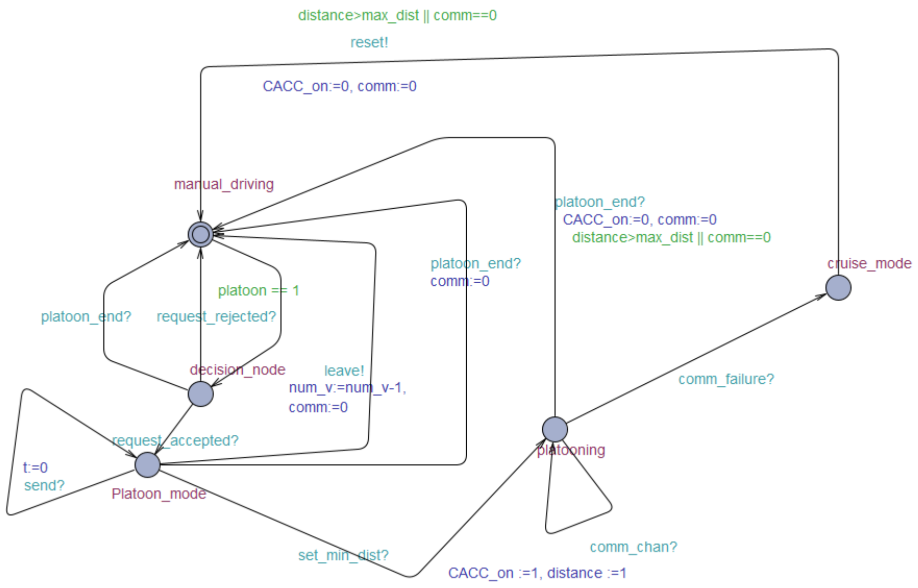


Fig. 15: UPPAAL model- Following truck

3.2 Maintaining constant distance [Tasawar Siddiquy]

The vehicle needs to adhere to a series of instructions in order to keep a consistent distance. To interact with the real environment, the vehicles will be fitted with a variety of sensors.

Figure 16 depicts the leading vehicle's control behavior. For instance, a safe space between trucks has been set at 30 meters, but this will change depending on future road conditions and the posted speed limit.

We made clear vehicle's control behavior in Figure 17, which will be the same as the leading vehicle's conduct with regard to maintaining distance. If one of the following vehicles disconnects from the platoon in an emergency, the following vehicle will be equipped with such a control system so that it may drive on its own without the leading vehicle's supervision.

A full view of the total system in UPPAAL can be viewed in our github repository and a verified simulation, to ensure there is no deadlock, can be seen in the following link: [4]

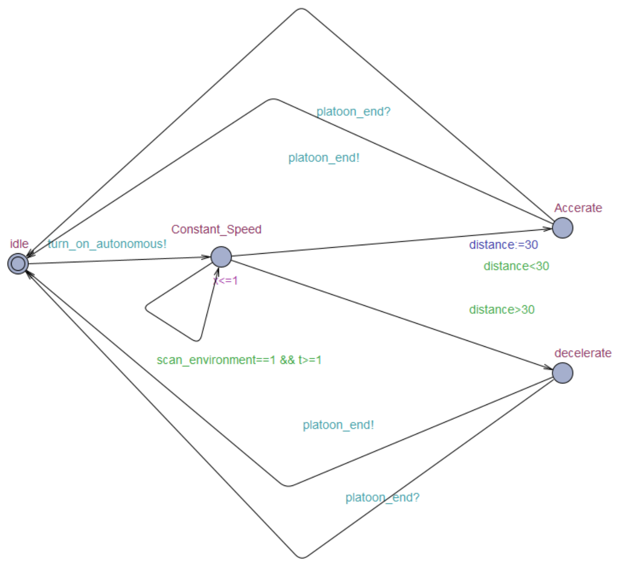


Fig. 16: Control Behavior of leading vehicle

4 Integrating Machine Learning Algorithm [Tasawar Siddiquy]

The machine learning algorithm is a subject that is heavily debated nowadays. It benefits autonomous systems a lot. The technology can make decisions akin to humans thanks to machine learning techniques. The most important aspect of platooning, determining our leading vehicle, was done using a machine learning system. After attempting to implement many alternative machine learning algorithms, we ultimately chose to adopt the supervised machine learning technique known as "decision Tree."

We used the Scikit library in python to implement the decision tree. The most important part of a machine learning algorithm is the data set and amount of data. In Figure 18, we chose some important features for our model. Firstly the route and the remaining distance to travel should be matched otherwise there is no benefit of platooning. We used the GPS coordinates to determine the destination of the vehicles. The body characteristic is also very meaningful for our platoon. If the leading vehicle is longer or bigger in size so it will be equipped with more sensors so it will be more suitable to become the leading vehicle. The leading vehicle should have also sufficient fuel so that’s also a feature of our data-set.

We used 20 percent data for testing and 80 percent data to train our model. Later on, more data can be added to increase the efficiency of our model. In Figure 19, after implementing the decision tree we checked the accuracy of our model. we got accuracy ’1’ because we used very fewer data. The very last column was our result means the possibility to be a

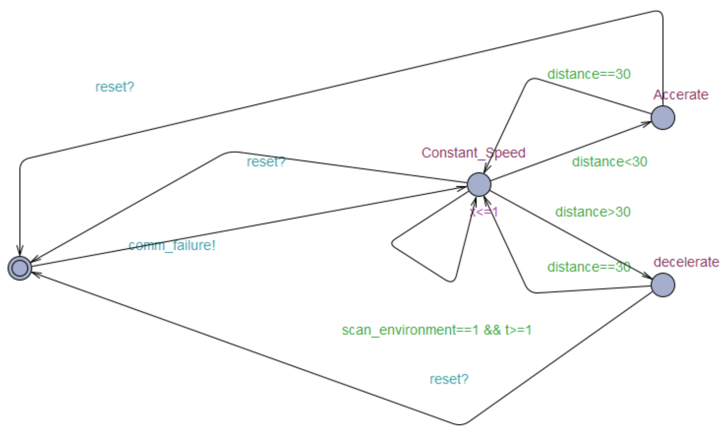
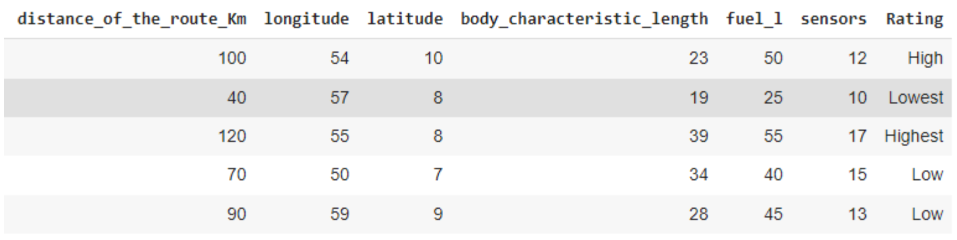


Fig. 17: Control Behavior of following vehicle



distance_of_the_route_Km	longitude	latitude	body_characteristic_length	fuel_l	sensors	Rating
100	54	10	23	50	12	High
40	57	8	19	25	10	Lowest
120	55	8	39	55	17	Highest
70	50	7	34	40	15	Low
90	59	9	28	45	13	Low

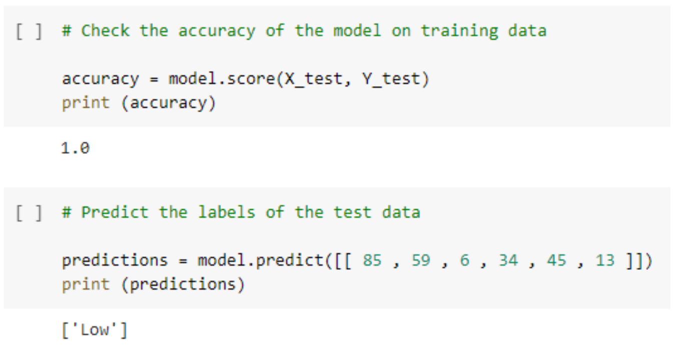
Fig. 18: Features

leading truck. We also predicted one of the rows to check our model. We got the correct decision after predicting.

To verify our model we used four different rows to let the model decide whether it's eligible to be the leading truck or not. In Figure 20, We selected some random rows from our data set to test our model. After running it we obtained the correct decision from our model.

5 Implementation and Simulation [Asm Nurussafa]

For the overall implementation and simulation of our system, we decided on using the simulator *SUMO* (Simulation of Urban Mobility) for the simulation. With this, we can benefit from the *TraCI* library, to control each of the vehicle as we will and also use the *Simpla*, to help us implement the platooning in vehicles. Together, using a python script, we can implement all of these, including our machine-learning algorithm discussed in section 4.



```
[ ] # Check the accuracy of the model on training data

    accuracy = model.score(X_test, Y_test)
    print (accuracy)

    1.0

[ ] # Predict the labels of the test data

    predictions = model.predict([[ 85 , 59 , 6 , 34 , 45 , 13 ]])
    print (predictions)

    ['Low']
```

Fig. 19: Prediction

SUMO is a free and open source traffic simulation suite. It is available since 2001 and allows modelling of intermodal traffic systems - including road vehicles, public transport and pedestrians. Included with SUMO is a wealth of supporting tools which automate core tasks for the creation, the execution and evaluation of traffic simulations, such as network import, route calculations, visualization and emission calculation. SUMO can be enhanced with custom models and provides various APIs to remotely control the simulation. [1]

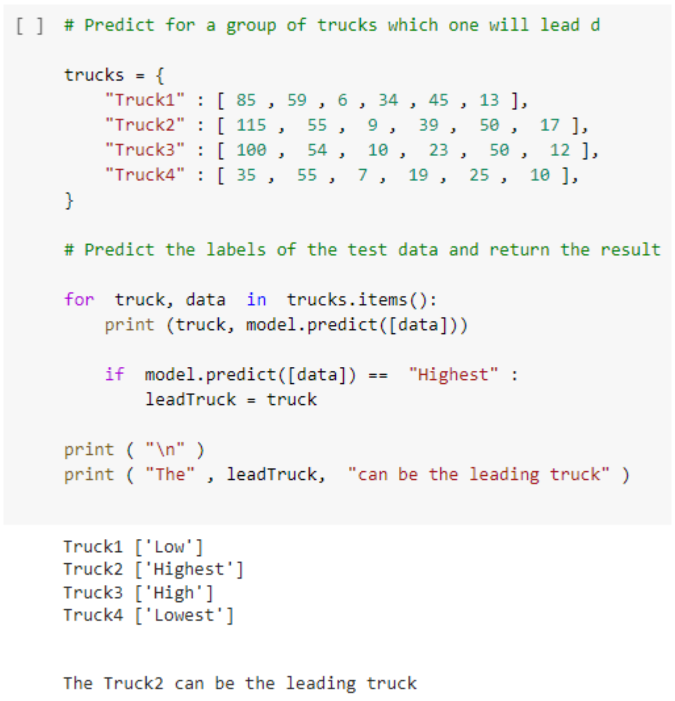
TraCI is the short term for "Traffic Control Interface". Giving access to a running road traffic simulation, it allows to retrieve values of simulated objects and to manipulate their behavior ön-line". [2]

Simpla is a configurable, platooning plugin for the TraCI Python client. It manages the spontaneous formation of vehicle platoons and allows you to define specific behavior for vehicles that travel inside a platoon. This is achieved by defining additional vehicle types which represent platooning modes, and by providing information, which type (mode) to use for normal, solitary travel, and which one to use when platooning is activated. Further, ’catch-up’ modes may be specified, which are used when a potential platooning partner is farther ahead downstream. [3]

We will now discuss a step-by-step method about how implemented our system.

5.1 Generating a map

Our idea is to create very simple map to show how the trucks behave before and after platooning. To do this, we first need to create a *.net.xml* file. We have used the *NET-EDIT* software to create a simple map. Here, we define the junctions and edges of the map through which the trucks will travel. This file can be viewed under our github repository.



```
[ ]  # Predict for a group of trucks which one will lead d

     trucks = {
         "Truck1" : [ 85 , 59 , 6 , 34 , 45 , 13 ],
         "Truck2" : [ 115 ,  55 ,  9 ,  39 ,  50 ,  17 ],
         "Truck3" : [ 100 ,  54 ,  10 ,  23 ,  50 ,  12 ],
         "Truck4" : [ 35 ,  55 ,  7 ,  19 ,  25 ,  10 ],
     }

     # Predict the labels of the test data and return the result

     for  truck, data  in  trucks.items():
         print (truck, model.predict([data]))

         if  model.predict([data]) ==  "Highest" :
             leadTruck = truck

     print ( "\n" )
     print ( "The" , leadTruck,  "can be the leading truck" )
```

```
Truck1 ['Low']
Truck2 ['Highest']
Truck3 ['High']
Truck4 ['Lowest']


The Truck2 can be the leading truck
```

Fig. 20: Testing and verifying

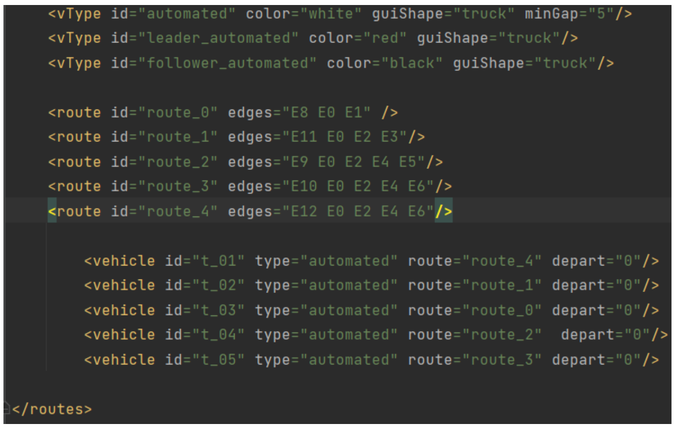
5.2 Generating a router file

The router file, *.rou.xml* file, is where we describe the route that each of our trucks is going to take. Here, we define the type of vehicles we wish to simulate, the routes taken (by defining the edges) and assigning these routes to the vehicle.

For our system and to keep it simple, we generated 5 trucks, starting from 5 different initial positions. All of the trucks have different routes and different final destinations but start at the same time. A snippet is shown below from the file: fig. 21

5.3 Configuration

The configuration file, *.sumocfg* file, is where define the input files and the processing files for our sytem. For our simple, we kept it simple and have mainly two input files, the *platoon.rou.xml* file, and the *platoon.net.xml* and a processing method. A snippet from the file is shown in fig. 22



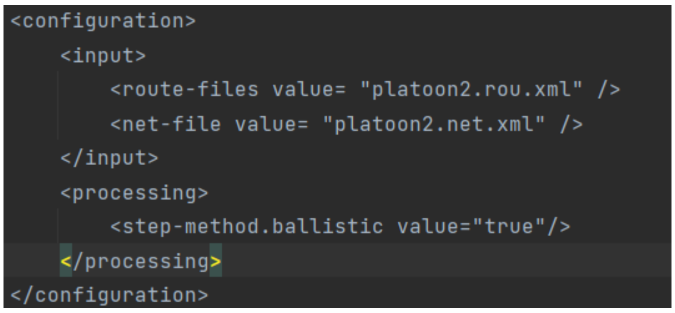
```xml
    <vType id="automated" color="white" guiShape="truck" minGap="5"/>
    <vType id="leader_automated" color="red" guiShape="truck"/>
    <vType id="follower_automated" color="black" guiShape="truck"/>

    <route id="route_0" edges="E8 E0 E1" />
    <route id="route_1" edges="E11 E0 E2 E3"/>
    <route id="route_2" edges="E9 E0 E2 E4 E5"/>
    <route id="route_3" edges="E10 E0 E2 E4 E6"/>
    <route id="route_4" edges="E12 E0 E2 E4 E6"/>

        <vehicle id="t_01" type="automated" route="route_4" depart="0"/>
        <vehicle id="t_02" type="automated" route="route_1" depart="0"/>
        <vehicle id="t_03" type="automated" route="route_0" depart="0"/>
        <vehicle id="t_04" type="automated" route="route_2"  depart="0"/>
        <vehicle id="t_05" type="automated" route="route_3" depart="0"/>

</routes>
```

Fig. 21: Router file (.rou.xml)



```xml
<configuration>
    <input>
        <route-files value= "platoon2.rou.xml" />
        <net-file value= "platoon2.net.xml" />
    </input>
    <processing>
        <step-method.ballistic value="true"/>
    </processing>
</configuration>
```

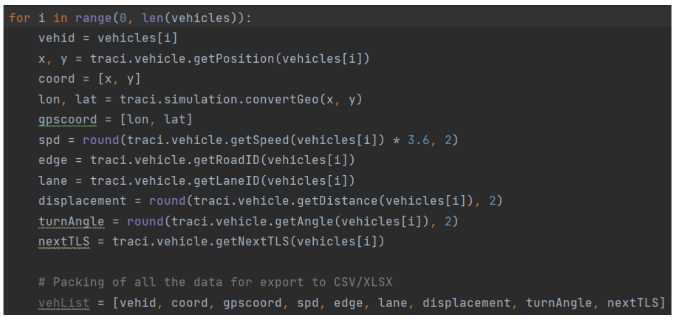
Fig. 22: Config file (.sumocfg.xml)

5.4 Python script using TraCI

After that, to be able to control each of the vehicle as we will, we need to create a python script. First, we include the necessary libraries and everything for the *traci* to initiate. After we have started the *traci.start()* we have a main *run()* function. So, for each of the simulation step we can gather information about our trucks, as for instance, the gps co-ordinates, speed , next traffic light signal etc. A snippet from the *run()* function is shown below in fig. 27

The output from this function can be integrated in our machine learning algorithm previously discussed, and that would help refine our platooning system. This can be seen fully in the python script file.

A video demonstration before enforcing platooning can be found here: [5]



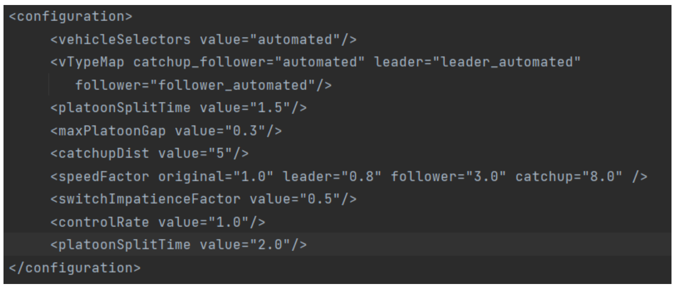
```python
for i in range(0, len(vehicles)):
    vehid = vehicles[i]
    x, y = traci.vehicle.getPosition(vehicles[i])
    coord = [x, y]
    lon, lat = traci.simulation.convertGeo(x, y)
    gpscoord = [lon, lat]
    spd = round(traci.vehicle.getSpeed(vehicles[i]) * 3.6, 2)
    edge = traci.vehicle.getRoadID(vehicles[i])
    lane = traci.vehicle.getLaneID(vehicles[i])
    displacement = round(traci.vehicle.getDistance(vehicles[i]), 2)
    turnAngle = round(traci.vehicle.getAngle(vehicles[i]), 2)
    nextTLS = traci.vehicle.getNextTLS(vehicles[i])

    # Packing of all the data for export to CSV/XLSX
    vehList = [vehid, coord, gpscoord, spd, edge, lane, displacement, turnAngle, nextTLS]
```

Fig. 23: Main run function in python script.

5.5 Implementing Platooning using Simpla

Using Simpla is very beneficial in the sense that has pre-built functions that can handle the platooning. Functions like *platooning manager, initialization* etc. are provided by the simpla library. To use these, first we need to create a configuration file for simpla that we want to implement. Here, we define the key parameters for our platooning scenario, for example, *platoonSplitTime*, *maxPlatoonGap* etc. A snippet from the simpla configuration file is shown in figure. 24



```xml
<configuration>
     <vehicleSelectors value="automated"/>
     <vTypeMap catchup_follower="automated" leader="leader_automated"
        follower="follower_automated"/>
     <platoonSplitTime value="1.5"/>
     <maxPlatoonGap value="0.3"/>
     <catchupDist value="5"/>
     <speedFactor original="1.0" leader="0.8" follower="3.0" catchup="8.0" />
     <switchImpatienceFactor value="0.5"/>
     <controlRate value="1.0"/>
     <platoonSplitTime value="2.0"/>
</configuration>
```

Fig. 24: Simpla configuration file.

Next, we import the Simpla library in our Python script and load our simpla configuration file using the *simpla.load()* function.

A video demonstration after enforcing the platooning can be found here: [7]

6 Conclusion

Throughout the semester, we have rather enjoyed working on such a project. Starting with modelling our system using SysML diagrams and then, diving into more depth by adding timing constraints to these ideas using UPPAAL modelling system. Next, we learnt how to implement different machine learning algorithms to realise such scenarios, e.g. determining the leading vehicle in the platooning system. After that, we experimented with different simulation tools. For our simulation, we used the tool SUMO and python scripts to realise and visual our system.

For further works of this project, we can work more on adding more datas into our machine learning algorithm, because at the moment, we have an accuracy of 1.0, which in real-lilfe scenario does not make sense and can never be implemented. Next, we need to work more on developing a better code to integrate our machine learning algorithms into our python script. Lastly, we should be able to refine our python script further to ensure the constant spacing between platooning vehicles and that platooning is actually ensured to be able to implement in real life scenarios.

Our GitHub repository can be found under: `https://github.com/arfatKamal/Team1-AutonomousTruckPlatooning`

10.07.22 & Lippstadt - Arfat Kamal

Bibliography

[1] About eclipse sumo. Eclipse SUMO - Simulation of Urban MObility. (n.d.). Retrieved July 10, 2022, from `https://www.eclipse.org/sumo/about/`

[2] TraCI. Traci - Sumo Documentation. (n.d.). Retrieved July 10, 2022, from `https://sumo.dlr.de/docs/TraCI.html`

[3] Intro. Simpla - SUMO Documentation. (n.d.). Retrieved July 10, 2022, from `https://sumo.dlr.de/docs/Simpla.html`

[4] Timed Automata Simulation– `https://drive.google.com/file/d/1Vj0xV88bzoWj8BBXbbfxXshG83xTPbnz/view?resourcekey`

[5] Video demonstration in Sumo before platooning– `https://drive.google.com/file/d/1y4f0axFk0qXUjun1LrOt11icx9Fd6uXH/view?resourcekey`

[6] Video demonstration in Sumo after platooning– `https://drive.google.com/file/d/1Ts-O4UjbIvQpmxLp1Gr3GV_TVHypowO4/view?resourcekey`

[7] Our GitHub link `https://github.com/arfatKamal/Team1-AutonomousTruckPlatooning`

7 Appendix

7.1 Declaration of Originality [Asm Nurussafa]

I, Asm Nurussafa, herewith declare that I 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. I agree that my work may be checked by a plagiarism checker.

10.07.22 & Hamm - Asm Nurussafa

7.2 Declaration of Originality [Tasawar Siddiquy]

I,Tasawar Siddiquy, herewith declare that I 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. I agree that my work may be checked by a plagiarism checker.

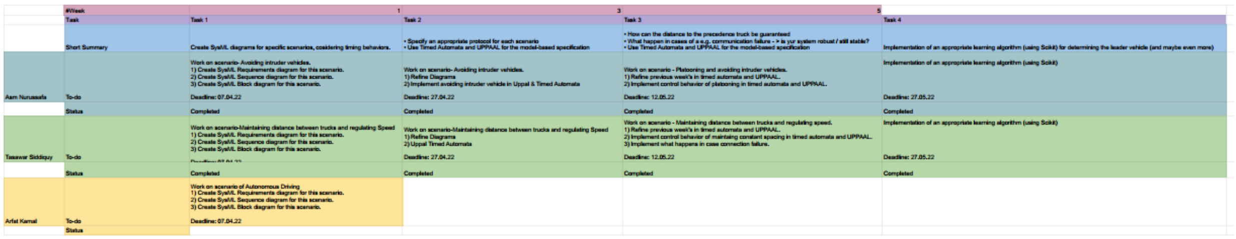
10.07.22 & Hamm- Tasawar Siddiquy

7.3 Declaration of Originality [Arfat Kamal]

I, Arfat Kamal, herewith declare that I 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. I agree that my work may be checked by a plagiarism checker.

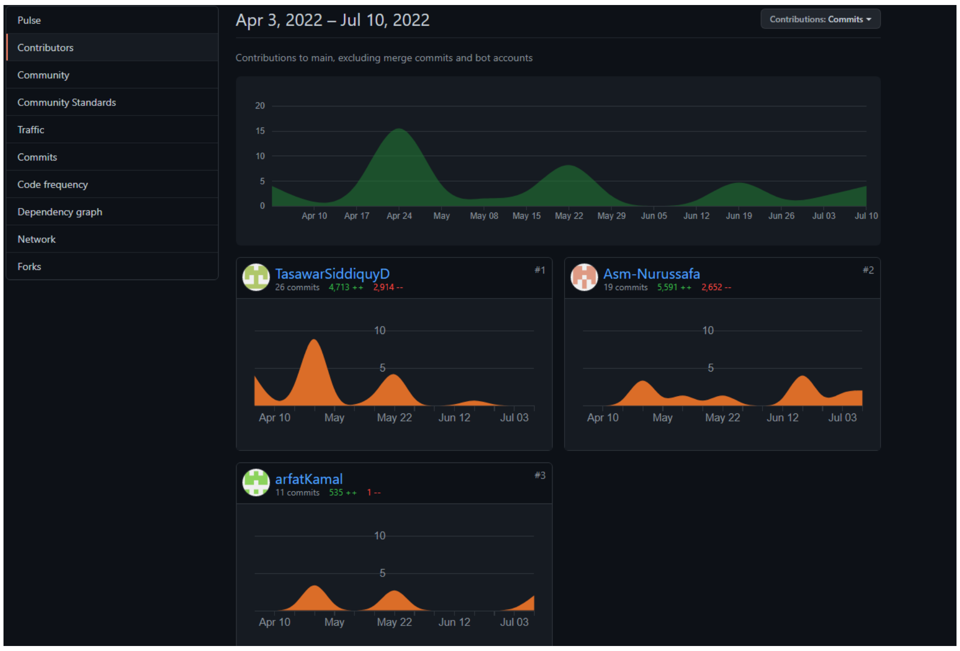
7.4 Task Responsibilities

7.5 History of GitHub upload



	#Week	1	3	5	
	Task	Task 1	Task 2	Task 3	Task 4
	Short Summary	Create SysML diagrams for specific scenarios, cosidering timing behaviors.	• Specify an appropriate protocol for each scenario • Use Timed Automata and UPPAAL for the model-based specification	• How can the distance to the precedence truck be guaranteed • What happen in cases of a e.g. communication failure - > is yur system robust / still stable? • Use Timed Automata and UPPAAL for the model-based specification	Implementation of an appropriate learning algorithm (using Scikit) for determining the leader vehicle (and maybe even more)
Asm Nurussafa	To-do	Work on scenario- Avoiding intruder vehicles. 1) Create SysML Requirements diagram for this scenario. 2) Create SysML Sequence diagram for this scenario. 3) Create SysML Block diagram for this scenario. Deadline: 07.04.22	Work on scenario- Avoiding intruder vehicles. 1) Refine Diagrams 2) Implement avoiding intruder vehicle in Uppal & Timed Automata Deadline: 27.04.22	Work on scenario - Platooning and avoiding intruder vehicles. 1) Refine previous week's in timed automata and UPPAAL. 2) Implement control behavior of platooning in timed automata and UPPAAL. Deadline: 12.05.22	Implementation of an appropriate learning algorithm (using Scikit) Deadline: 27.05.22
	Status	Completed	Completed	Completed	Completed
Tasawar Siddiquy	To-do	Work on scenario-Maintaining distance between trucks and regulating Speed 1) Create SysML Requirements diagram for this scenario. 2) Create SysML Sequence diagram for this scenario. 3) Create SysML Block diagram for this scenario. Deadline: 07.04.22	Work on scenario-Maintaining distance between trucks and regulating Speed 1) Refine Diagrams 2) Uppal Timed Automata Deadline: 27.04.22	Work on scenario - Maintaining distance between trucks and regulating speed. 1) Refine previous week's in timed automata and UPPAAL. 2) Implement control behavior of maintaing constant spacing in timed automata and UPPAAL. 3) Implement what happens in case connection failure. Deadline: 12.05.22	Implementation of an appropriate learning algorithm (using Scikit) Deadline: 27.05.22
	Status	Completed	Completed	Completed	Completed
Arfat Kamal	To-do	Work on scenario of Autonomous Driving 1) Create SysML Requirements diagram for this scenario. 2) Create SysML Sequence diagram for this scenario. 3) Create SysML Block diagram for this scenario. Deadline: 07.04.22			
	Status				

Fig. 25: Task Responsibilities



Apr 3, 2022 – Jul 10, 2022

Contributions to main, excluding merge commits and bot accounts

TasawarSiddiquyD 26 commits 4,713 ++ 2,914 --

Asm-Nurussafa 19 commits 5,591 ++ 2,652 --

arfatKamal 11 commits 535 ++ 1 --

Fig. 26: Git Contributors

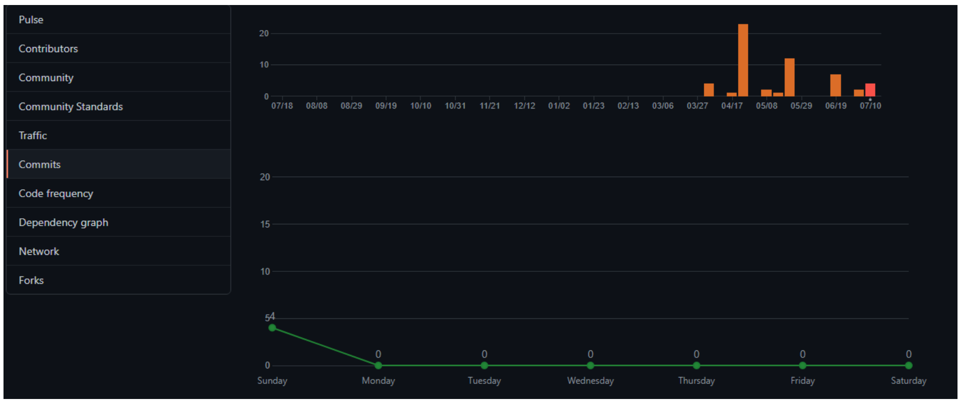


Fig. 27: Git Commits