UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

Vehicular Networks and Cooperative Driving
A.A. 2022-2023
Proff. Renato Lo Cigno
Lorenzo Ghiro

Implementing a Virtual Safety Car in Veins

Cavalletto Leonardo
Reboldi Mattia

Executive Summary

This project implements a Virtual Safety Car in a Speed Circuit during a race.
The race is simulated by spawning an arbitrary number of cars on all three lanes.
At a determined time an accident is forced, so the vehicle ”crashes” and stops in the outer lane for a set amount of time. When that vehicle has stopped, it sends messages to the cars around and to the closest RSU. When the RSU receives a message, it retransmits it to the surrounding RSUs, reaching all the cars on the circuit. Once a car receives the message, it moves to the inner lane and decreases its speed.
When the duration of the accident is over, the car is removed from the circuit and the other vehicles will continue the race without the speed limits.

Contents

1 Project Description

This project implements a Virtual Safety Car in a Speed Circuit during a race. The scenario is the following:

- The circuit is a ring with three lanes generated with a python script.
- We chose a number of twenty vehicles, each with his proper list of attributes, split on all three lanes.
- Once spawned, vehicles will start the race increasing their speed to their max and they are free to overtake each other.
- At a determined time an accident is forced, so a chosen vehicle stops in the outer lane. When the crashed car is fully stopped, it starts sending messages every two seconds to the surrounding cars and RSUs.
- Once the message has been received, the other vehicles move to the inner lane and decrease the velocity to 55 m/s.
- The cars continue in this sort of ”virtual safety car” scenario until they receive the last message from the crashed car so the accident is over.
- The crashed car is finally removed from the circuit and race can continue without speed and lane limits.

2 Experiment Design

The project doesn't follow a specific protocol. The accident is predetermined and forced at a specific time. Each car simply executes a list of commands when a message from the crashed car is received. The exact response of the cars during the executions of those commands, such as braking, accelerating or changing lane, is dependant to some SUMO parameters and so it is not always precisely predictable.
There are some parameters that can be arbitrarily modified both in SUMO and in VEINS:

- Number of cars: the number chose is twenty according to the F1 races but it can be simply modified in the file rou.xml. The number of the vehicles spawned on each lane can be modified too.
- startAccident: This parameter in the .ini file indicates when the accident starts. If startAccident is low, the accident will happen early in the simulation and the cars will be really close one another otherwise, if the value is high, the accident will occur later in the simulation so the cars will be spread along the circuit and they have more space to enter the inner lane. The chosen value is forty.
- accidentDuration: another .ini parameter is the accidentDuration that defines the time in seconds in which the crash vehicle stays in the outer lane. It's better to choose a higher value because, if it is too small, not every car could change lane and reduce their speed so the Virtual Safety Car is not uniformly applied to everyone.
- Crashed Car: In the .ini file it's possible to modify which car is involved in the accident changing the number of the node. This parameter is less important than the others so the choice is free and it doesn't change in anyway the result of the simulation. In the simulation the designated car is the car with index 5.

2.1 Implementation

The project's implementation consists in edits of SUMO files and customization of class files from a laboratory example in VEINS.

2.1.1 Ring Generation

In order to create the circuit, it has been used a Python script named ringGen.py in which the requested parameters are:

- lanes: The number of lanes in the circuit. For this project the ring presents three lanes because there is more room for maneuvers and the third one helps the implementation of the accident.
- dimension: Total length of the ring in meters. The chosen length is 4000 m as a short/medium F1 circuit length.
- speed: Maximum speed allowed in m/s. The chosen value is 100 m/s and it can't be reached by the speed of the vehicles because in a race scenario the cars will not have speed limits.
- edges: Number of edges. It's been chosen a high value of edges to get a more round circuit.

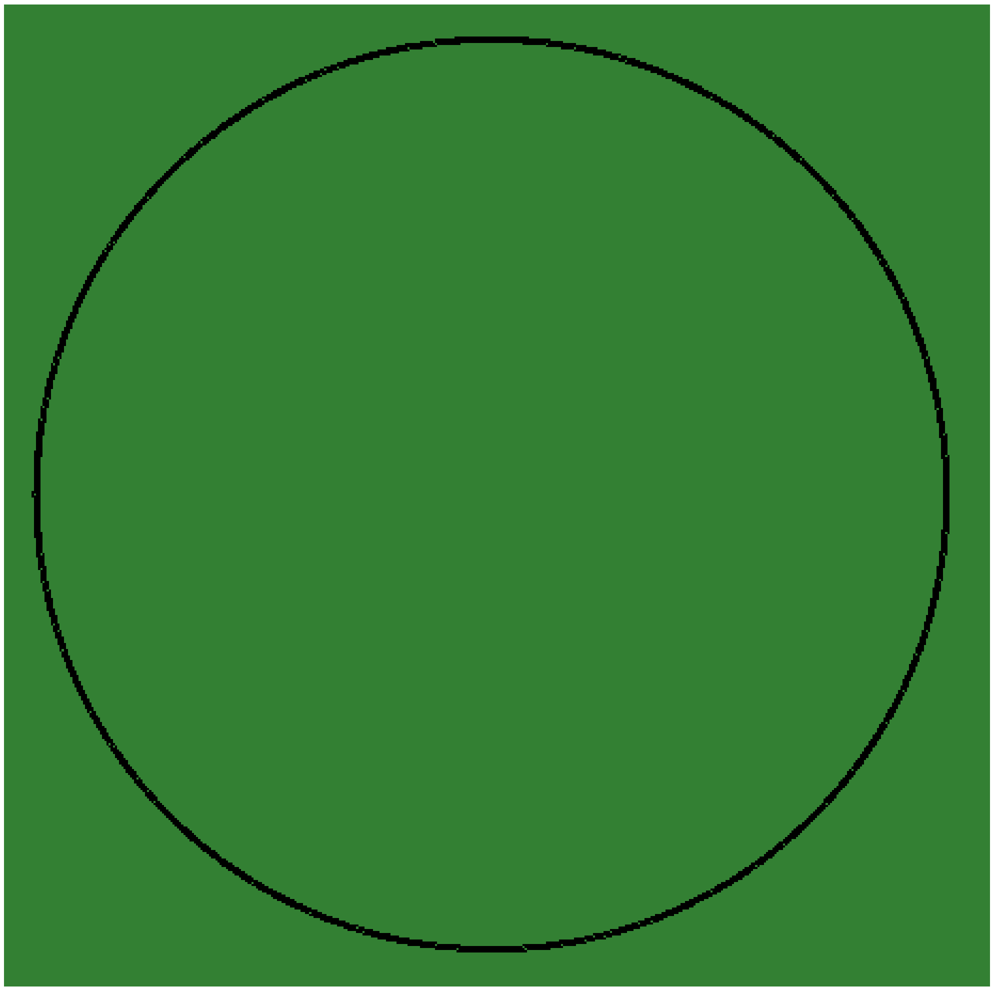


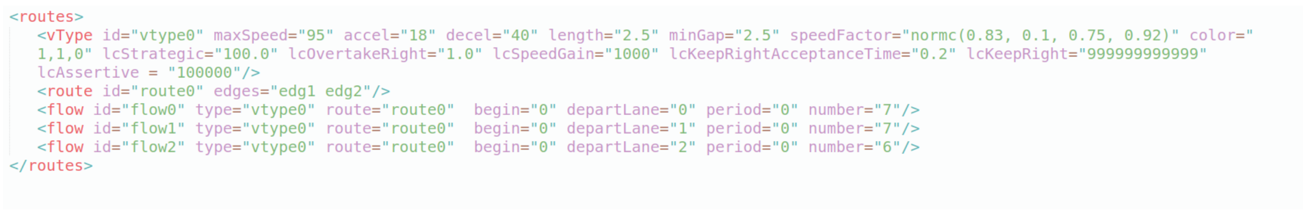
Figure 1: Generated circuit of the simulation

2.1.2 ini file

The playground is set to a square with edge equal to 1500 meters. The decided simulation time is 200s. Once the race resumes after the accident, there is no need to look at the system anymore because the goal of the project is to observe the reaction of the cars at the accident. The Antennas used for cars and RSUs are monopoles situated at the height of 3 m for the RSUs and 1.895 m for cars.

2.1.3 Vehicle configuration in SUMO

Here's a screenshot of the vehicle configuration with some added and modified parameters in the rou.xml file:



```xml
<routes>
   <vType id="vtype0" maxSpeed="95" accel="18" decel="40" length="2.5" minGap="2.5" speedFactor="normc(0.83, 0.1, 0.75, 0.92)" color="
   1,1,0" lcStrategic="100.0" lcOvertakeRight="1.0" lcSpeedGain="1000" lcKeepRightAcceptanceTime="0.2" lcKeepRight="999999999999"
   lcAssertive = "100000"/>
   <route id="route0" edges="edg1 edg2"/>
   <flow id="flow0" type="vtype0" route="route0"  begin="0" departLane="0" period="0" number="7"/>
   <flow id="flow1" type="vtype0" route="route0"  begin="0" departLane="1" period="0" number="7"/>
   <flow id="flow2" type="vtype0" route="route0"  begin="0" departLane="2" period="0" number="6"/>
</routes>
```

Figure 2: Parameters in Rou File

- *maxSpeed*: The vehicle's (technical) maximum velocity (in m/s)
- *accel*: The acceleration ability of vehicles of this type
- *decel*: The deceleration ability of vehicles of this type
- *length*: The vehicle's netto-length

(These chosen parameter values are inspired on real F1 cars)

- *minGap*: Empty space after leader
- *speedFactor*: The vehicles expected multiplier for lane speed limits and desiredMaxSpeed. Having a distribution of speed factors is beneficial to the realism of a simulation. In this project has been chosen a truncated normal distribution defined as shown:
  speedFactor = "normc(mean, deviation, lowerCutOff, upperCutOff)"
- *lcStrategic*:The eagerness for performing strategic lane changing. Higher values result in earlier lane-changing.
- *lcOvertakeRight*: The probability for violating rules gainst overtaking on the right. In a race scenario the overtakes are allowed also from the right.
- *lcSpeedGain*: The eagerness for performing lane changing to gain speed. Higher values result in more lane-changing.
- *lcKeepRightAcceptanceTime*: Time threshold for changing the willingness to change right. The value is compared against the anticipated time of unobstructed driving on the right. Lower values will encourage keepRight changes.
- *lcKeepRight*: The eagerness for following the obligation to keep right. Higher values result in earlier lane-changing. This parameter has been chosen so a vehicle can drive following the ideal trajectory.
- *lcAssertive*: Willingness to accept lower front and rear gaps on the target lane.

Since SUMO has been designed for urban scenarios with traffic rules to follow, these parameters aim to improve the realism of a car race simulation, trying to ignore some of the rules such as the overtaking from the right.
In addition, the parameters managing the flow of vehicles are the following:

- departLane: Determines on which lane the vehicle is tried to be inserted
- period: the amount of time between two consecutive spawns in the same lane given in seconds.

The goal is to simulate a launch starting grid as in a Nascar race.

2.1.4 TraCIDemo11p.msg

The messages are composed of these parameters:

- demoData: it's the road's iD where the accident happened
- senderAddress: it identifies the vehicle which is sending the message. In this project the sender is the crashed car.
- TTL: (time to leave) it's a number decreased every time once retransmitted. In this case, the chosen value is zero so that each car that receives a message doesn't repropagate and the retransmission is entrusted only to the RSUs.
- timestamp: the time of the simulation when the message has been sent.

2.1.5 RSU

The RSUs in the project are 4 and they are located inside the circuit creating a square centered in the circuit with an edge of 424.3 meters. The transmission power of a RSU is 20mW as the power of the cars. The chosen max interference distance is 480 meters so that a vehicle can reach at least one RSU anywhere along the circuit and a RSU needs to send the message to the nearest RSUs to propagate everywhere in the circuit. The delayed message is sent after a period of time uniformly distributed between 0.1 and 0.2 seconds, so that the complete retransmission of the message over the entire circuit happens before that the crashed car sends the next message. The RSU completely ignores the TTL. When a message is transmitted to a RSU, there is a check between it and the set of previously received messages. If there is no message with the same timestamp, the RSU repropagates the message and it updates the set adding the new message. Instead, if the message was already in the set of the received message, the RSU does not propagate it. This procedure can be skipped simply putting the avoidDuplicates boolean to false. As there are four RSUs, the reason of the true avoidDuplicates is to prevent infinite retransmissions between RSUs.

2.1.6 Handling SelfMessages

To force an accident the car which will crash needs some self messages as a way to know when to start the accident and how much it will last. The selfmessages are managed in TraciMobility.cc. In the initialize method of this class the selfmessages startAccident and stopAccident are created. The startAccident selfmessage is scheduled at simTime() + accidentStart, in which simTime() is almost at time zero and accidentStart is defined in the .ini file. When the startAccident selfmessage is received the car will be forced to change lane to the outer one and its speed will decrease to zero. Furthermore, the selfmessage stopAccident is scheduled at simTime() + accidentDuration, in which accidentDuration is set in the .ini file. Once the stopAccident selfmessage is received the crashed car is removed from the circuit.

2.1.7 TraCIDemo11p

To implement the communication between the crashed car and the others in TraCIDemo11p.cc there is the handlePositionUpdate method which checks that, if a vehicle is stopped for at least two seconds (this is unusual in a race), it sends a message to the surrounding cars and RSUs. The message is a TraCIDemo11p.msg, and it is populated with the parameters listed in 2.1.4 paragraph. When a vehicle receives a message, the TTL is checked and if it is zero (as it is set by default in this project) there is no repropagation. Since also the crashed car receives the message about the accident from the RSUs, then that car must be excluded from the emergency maneuvers. To do that, before giving commands, the speed of the vehicle has to be checked because every car, except the crashed vehicle, has speed greater than zero. The emergency maneuvers consists of a forced change of lane to the inner one and an emergency braking that sets a constant speed of 55 m/s. When the accident is over the race will resume without any speed and lane limitations.

3 System Behavior

In this paragraph it's shown how the system behaves starting from the spawn of the vehicles until the end of the simulation. The cars spawn on each lane, three at the time, as closed as possible, starting from the grid.

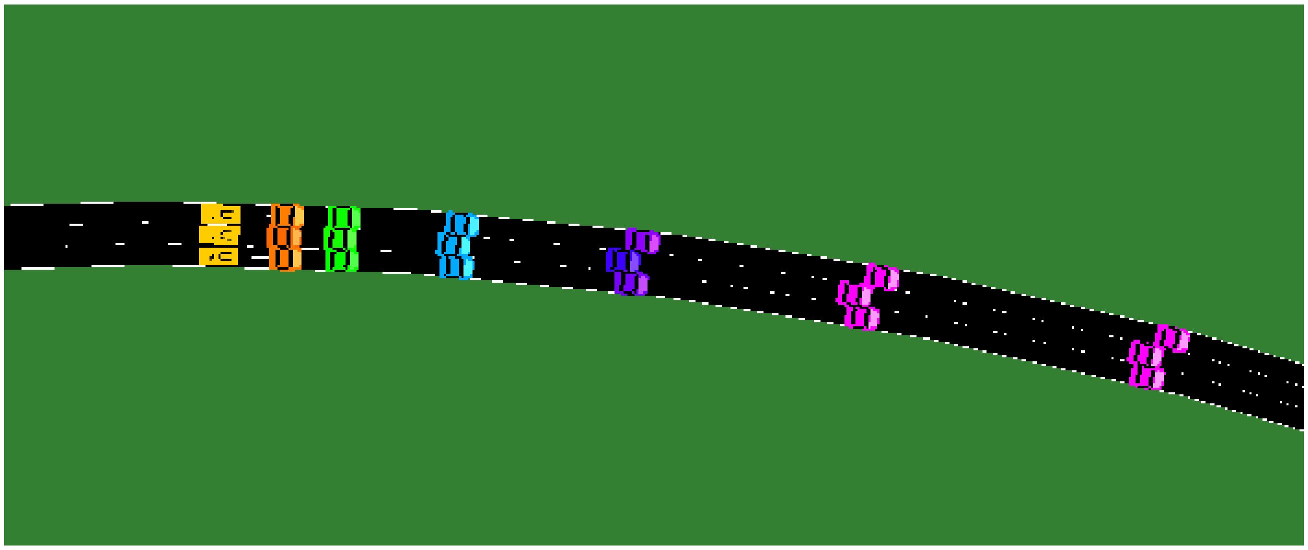


Figure 3: Spawn of the vehicles in the circuit

When the race starts, each vehicle begins to increase its speed until reaching the maximum and then they continue to maintain it as a steady speed. The vehicle colors in the pic represent different values of velocity from red (0 m/s) to violet (higher speeds). Due to the car parameters inserted in the rou file, each car has a different speed profile. The graphics below show different speed profiles in which the maximum speed is reached at different moments and the second car. In the first image the car is full throttle until it's steady velocity while the second presents a little deceleration due to a lost of position during a overtake.

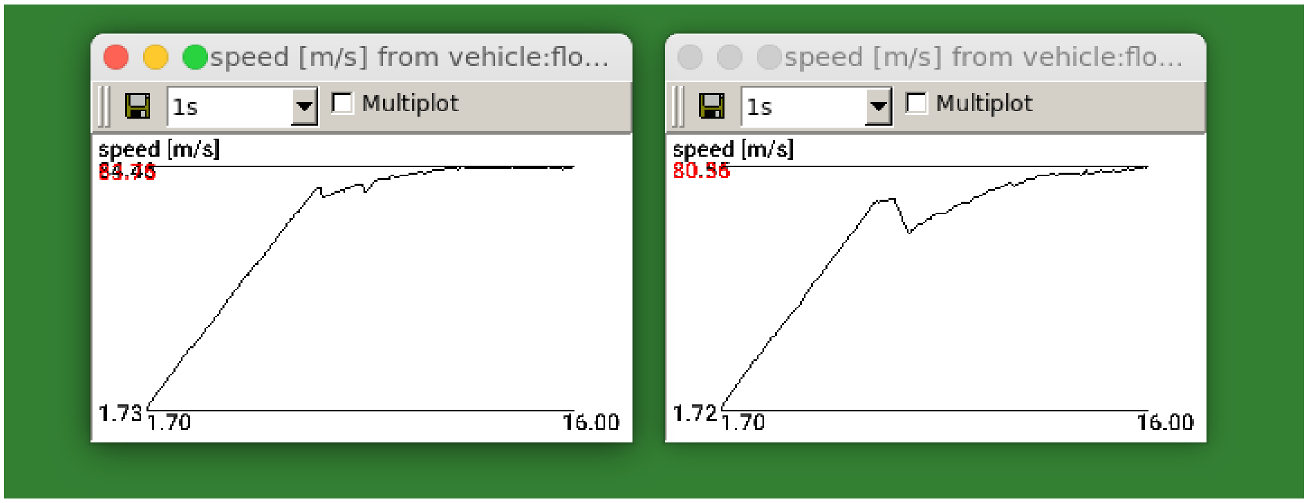


Figure 4: Comparison between different speed profiles

When the simulation time is equal to the startAccident time, the designated vehicle moves to the outer lane and starts to brake until its speed reaches the 0 m/s. The crashed vehicle turns red in the SUMO graphic interface.

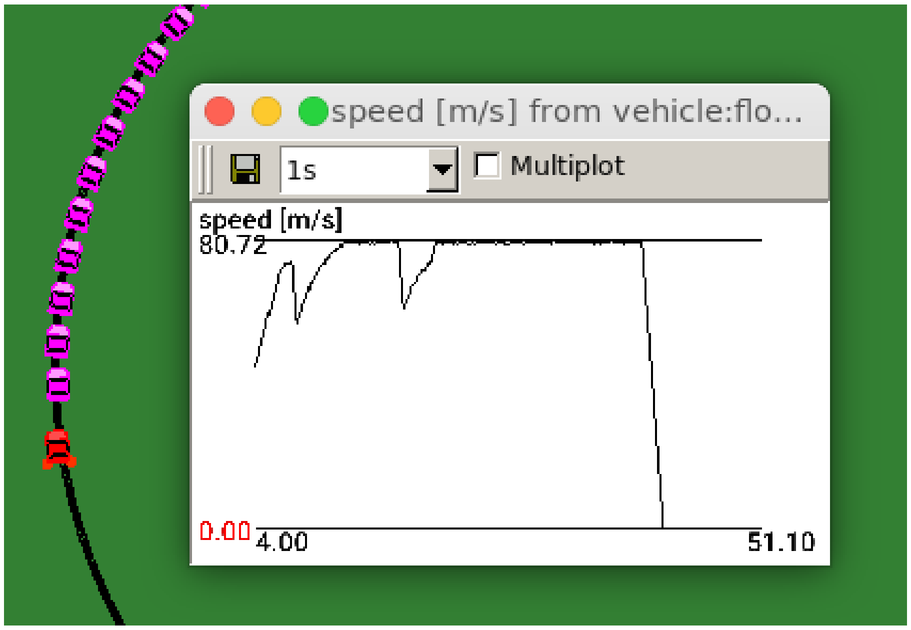


Figure 5: Speed profile of the crashed vehicle

Once stopped for at least two seconds, it starts to send TraciDemo11p message to surrounding cars and RSUs. The RSUs repropagates the message instead cars don't. When a car has received the message, its color turns green in Veins. The Veins graphics below show this communication:

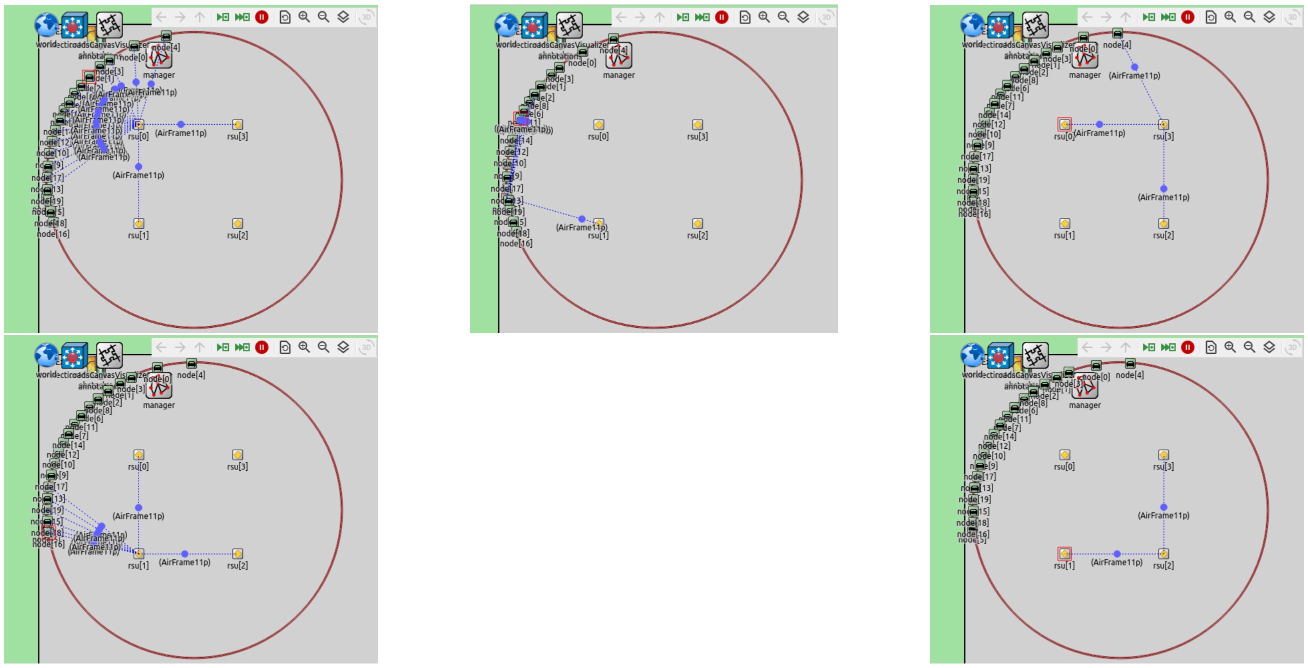


Figure 6: Propagation of a message

After receiving a message, the other cars will slow down to 55 m/s and move to the inner lane. They maintain this state of ”Virtual Safety car” until the crash is over and the vehicle is removed from the circuit. Since every cars have the same speed, they form a sort of ”uncooperative platoon”.

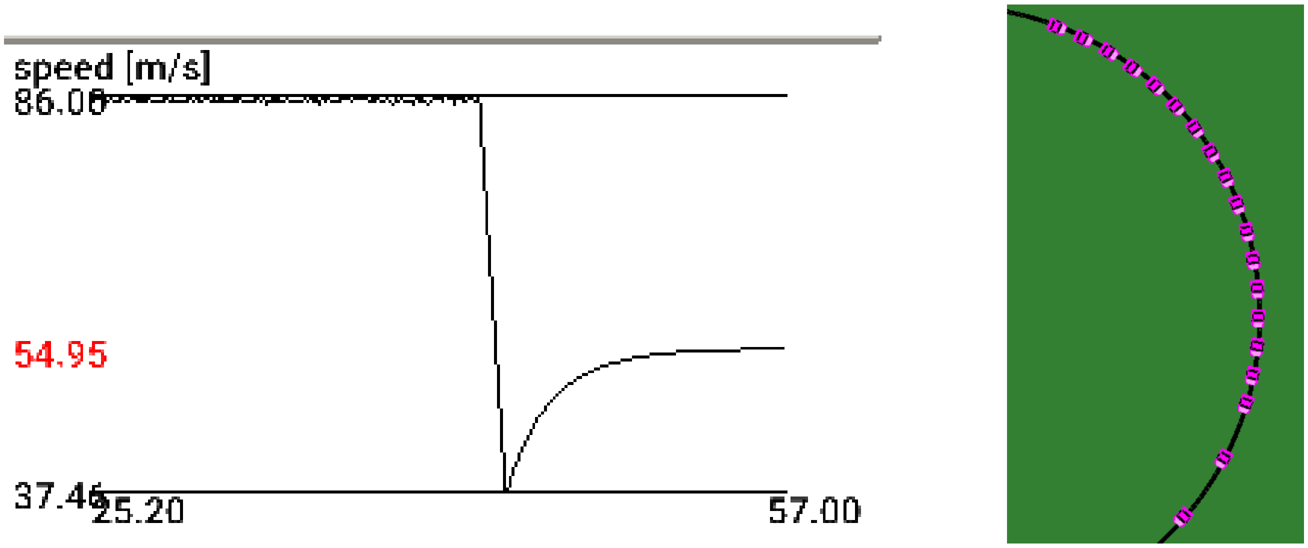


Figure 7: Braking in the Virtual Safety Car state

In the previous diagram, the minimum of the function represents a speed lower then 55 m/s to help the formation of the cars column, let the other cars get in it.

When the accident is over, the crash car is removed from the circuit and the other vehicles resume the race. The following graph describes an example of a complete speed profile of a random car in the simulation.

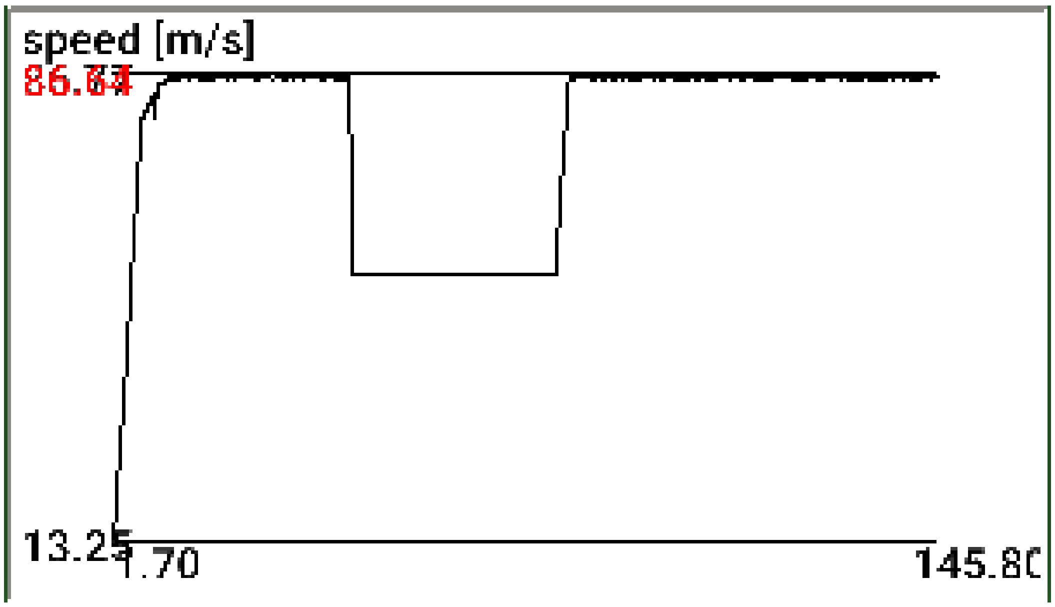


Figure 8: Speed profile of a car during the race

4 Conclusions and Future Work

This work shows how an emergency maneuver can be implemented in a race scenario using RSUs for retransmission of the emergency messages. Despite the car behavior during the emergency maneuvers is quite realistic after the reception of a message, this project presents some limitations. The biggest limitation is the resume of the race. In fact, every vehicles need to know from the beginning the startAccident and the accidentDuration values so that they already know when the accident is over, making the conditions unrealistic.
Many future works are possible obvious extension to this project:

- Implementing two different TraciDemo11p messages as StartAccident and StopAccident so that cars can react at the end of the accident without the knowledge of the accident duration.
- Changing the layout of the circuit so that it looks more like a real circuit and it's possible to see the behavior of the cars in the corners.
- Trying to implement this project on Plexe to manage a real car platoon using both human driven cars and platooning cars.

A Usage Report

Code available at: https://github.com/Leonardo-Cavalletto/veins/tree/main
This code is based on VEINS, and requires also an OMNeT++ and SUMO installation. Quoting the respective websites:

- Veins (car2x.org)) is an open-source framework for running vehicular network simulations extending OMNeT++ and SUMO to offer a comprehensive suite of models for IVC simulation.
- Eclipse SUMO - Simulation of Urban MObility) is an open source, highly portable, microscopic, and continuous multi-modal traffic simulation package designed to handle large networks.
- OMNeT++ Discrete Event Simulator (omnetpp.org) is an event-based network simulator.

To run the project it is required to download the branch vncd23 from the previous github link, then two terminals must be opened from the directory ”veins/examples/veins”. On one of them, it must be executed the following command: *veins_launchd -vv -c sumo-gui*
Then on the other terminal this command must be used: *./run -u Cmdenv -c Default -r 0*

CopyRight