Simulation of Traffic Flow to Analyze the Vehicle Behaviour on a Multi Lane Motorway using Simpy

Jaswinder Singh
MSc in Data Analytics
National College of Ireland
Dublin, Ireland
x19219997@student.ncirl.ie

***Abstract*—The main objective of the following study is to simulate the behaviour of different types of vehicles (electrical and diesel) on a two lane motorway using the discrete simulation python package called simpy.Various different models are constructed to simulate different aspects of the multi-lane motorway study. These models include *model for traffic generation* which includes calculation of throughput, average travelling times and speeds, *model for behaviour of human drivers* which includes various different methods like *chooseLane()* which enables a vehicle to choose the desired lane at random (but with specific weights given to each lane) and *model for vehicles* which incorporates the different kinds of vehicles on the road like *electrical*, *diesel*, etc. Each class of the vehicle has been given different minimum and the maximum accelerations based on the data collected from various surveys. Different simulation models were run and the results in the form of position time plots were reported. The conclusions from the study can help in designing and studying various aspects in a multi-lane traffic simulation and can also help in reducing the traffic congestion.**

***Keywords*—Multi-lane traffic simulation, Simpy, Discrete Simulation**

I. Introduction

With the increasing number of vehicles on the roads, motorways and highways, it has become necessary to manage the traffic congestion problem. According to some studies, we now manufacture more cars in a month than we used to in a year, a decade ago. The increasing number of vehicles have caused frequent road accidents leading to multiple causalities and losses to properties while also adding an additional pressure on the traffic management systems in the urban areas. According to the study conducted by Singh [1] on the investigation of different reasons of crashing, among the key factors or the causes of road accidents like vehicles, drivers, external environmental factors such as slick roads, weather, etc., failure of a vehicle's component, the majority of the accidents (almost about 94%) occurred because of the driver. Among these 94% accidents, 41% were due to cognitive errors, 33% were due to the mistake in the decision-making process by the driver and 11% due to the improper operation. Therefore, a fully automated self driving car that can replace human drivers has turned out as an important research subject in the future development in this domain.

The automated cars bring with themselves, a bundle of challenges to overcome as well. For example, harsh weather like heavy rains interferes with the sensors and that could trigger accidents. Another reason that self driving are still not on roads is because of shortage of the amount of training data available for the machine learning algorithms to learn from. Self driving cars are powered by machine learning algorithms which are hungry for data. They learn from thousands of hours of data of different real world scenarios on the roads and without it they are practically useless. There are various ethical issues associated with self driving cars as well like the well known 'Trolley Problem' which will require the program or AI(artificial intelligence) driving the car to take a decision involving ethical dilemma of whether to sacrifice one person to save many [2].

Even though we have a long way to go for having fully automated cars on the road, we have made tremendous progress in tackling the traffic congestion problems on the road. We can now simulate various aspects of the real world driving scenarios on roads and highways through different simulation softwares which can aid us greatly in solving various traffic congestion problems and also reduce (if not completely avoid) road accidents. One such real world scenario is the merging of lanes on a road or a motorway to reduce congestion.

A team of researchers at Trinity College Dublin (TCD) conducted a study in which they designed an algorithm allowing partially or fully automated vehicles to self organise and alleviate the effects of lane merging [3]. They called this motorway, the *"Self Organising Motorway"*. Self organisation can be though of as a process in which the elements in a system interact on their own and optimize their behaviour over time without any external authority imposing it. This same self organising nature can be seen in traffic control management at intersections to avoid the traffic jam problems [4]. The simulation software used in the motorway study is called *VISSIM*. The team at trinity developed a multi-modal simulation of traffic flow in a scenario in which a three lane motorway merges into two lanes after a distance of 2km. The primary objective of this study is to replicate different aspects of the *Self organising Motorway* research considering real world scenarios, using the discrete event simulation python package known as *simpy*. This study will focus on creating separate models for simulating different scenarios in a two lane motorway. These scenarios will also incorporate the erratic behaviour of the human drivers using different distributions for speeds and by also introducing the lane selection function

which allows an entering vehicle to choose between the two lanes at random. But since we are considering a real world scenario, we have given specific weights to each lane depending on the lane nature(slow/fast). Different simulation models will be constructed for each of scenarios and the results(throughput, average travelling time, average speed and traffic density) are reported. These models will be discussed in the section IV. The results obtained and their interpretations will be summarized in section IV.

II. Literature Review

Before conducting the study and developing different simulation models, it is pertinent to critically review the existing literature in the domain. In the following subsequent paragraphs, some of the key works in the domain of traffic simulation have been summarized.

In a review done by Azlan and Rohani [5], the different kinds of the traffic simulation models have been discussed. A traffic simulation model focuses mainly on three output values to tackle various traffic problems. These three output values are:

- *Traffic Flows*: For tacking problems based on congestion.
- *Network Element*: It contains the different elements of roads like link, merge, etc. which affect the overall geometry of the road.
- *Skim Category*: For tacking the issues like optimizing the cost and time for travel.

They also described three classes of traffic simulation models namely Microscopic, Macroscopic and Mesoscopic and the different sub classes involved as well. The one subclass relevant to our study is the Lane Changing Models which comes under the Microscopic class of models. It was originally proposed by Gipps [6]. The authors also discuss the various parameters involved in a lane changing model in great detail.

In a study conducted by Srikant et. al [7] to analyse the lane changing behaviour of vehicles on highways, different parameters like traffic volume, number of lanes have been studied to analyse their effect on the number of lane changes. They also quantified the maximum number of lane changes and the lane change at capacity level for four, six and eight lane divided highways. They used the VISSIM simulation software to generate and validate their simulated models. They found that traffic volume is correlated with the lane change data by the third degree polynomial trend. They tested and validated their model based on the field data.

In an interesting research done by Hu et. al [8], a deep learning algorithm based on reinforcement learning called policy gradient is used to create an end-to-end automated lane changing model. They employed a reinforcement learning algorithm primarily to incorporate the various unpredictable scenarios encountered in the real world, in lane changing situations. They used the acceleration and the steering wheel angle as the action space, the distance information about the lane boundary and the surrounding vehicles as state space and the avoiding of collision and setting different expected lane changing distances as the reward function.

Rodaro and Yeldan [9] used a discrete computation model called *cellular automata* to create a multi-lane traffic simulation. The advantage of using cellular automata for simulation processes is that it has high computational efficiency. The microscopic traffic models on the other hand, are comparatively more computationally expensive but they contain more accurate characteristics of vehicles and driver in the real world scenarios. Their proposed model has the computational efficiency of cellular automata and the accuracy of the microscopic models. The experimental evaluation of their proposed model showed that it was able to reproduce the behaviour of a typical traffic flow with heterogeneous traffic.

Harb et. al [10] conducted a simulation study comparing different schemes for three lane merging scenarios. They used VISSIM software to simulate a 2 to 1 work zone lane closure scenario for three different Maintenance of Traffic (MOT) plans. They collected the field data in order to validate and callibrate their simulation models. They analyzed the effectiveness of each of these MOTs for different driver compliance rates, traffic volumes and truck percentages.

Labib et. al [11] combined various data mining and simulation techniques to address the traffic congestion problem. They conducted a case study of signals at the intersections in Dhaka, a city in the country of Bangladesh. They studied the video footage of traffic from different intersections in the city and used a computer vision tool in order to extract the traffic flow data. After undergoing an additional further data mining process, they were able to achieve an accuracy of over 90%. They created and simulated two different hybrid scenarios at micro level. The results of their simulation helped in reducing the vehicle queue length by 40%, increasing the average travel speed and reducing the overall congestion in traffic.

The optimization of simulation parameters is an important part of any simulation study. The optimized parameters can then be further used to analyze and tackle the different real world scenarios. Sulejic et. al [12] conducted an important research in which they studied the optimization of a lane changing distribution for a motorway segment. They particularly focus on the problem of lane changing concentration caused by the increase in the traffic flow leading to congestion. They propose an algorithm for optimizing the distribution for lane-changing. They further evaluated their proposed algorithm on a one sided ramp motorway weaving segment having a weaving configuration of just 400 meters between the merging and diverging segments of the motorway.

III. Methodology

A well defined methodology is necessary in order to successfully implement any particular simulation model. Our objective is to simulate the traffic flow on a two lane motorway using the discrete simulation package called simpy in python. Various different concepts and characteristics of the object oriented programming(OOP) like encapsulation, inheritance were utilised in framing the methodology for the research. All the different elements and entities of the simulation were

constructed using objects and classes. The methodology used in this study can be broadly divided into the following parts:

A. Construction of Lanes (class Lane)

The lanes of the motorway were constructed in the *Lane* class. A lane can be constructed in two different ways. The first method is to attach small lane segments to each other to form a full lane. the second one is to use just a single long lane segment for the entire lane. We assigned the name 'slow lane' to the left lane of the motorway and 'fast lane' to the right lane. The left lane is used more frequently by vehicles than the right lane. Subsequent parallel lanes can be added to the left and the right of the current lane by *attachLeft()* and *attachRight()* functions in the Lane class. Similarly, no. of lane segments can be added to the right and left of the current lane by *widenRight()* and *widenLeft()* functions respectively. This class also defines the enter and the leave lane events for the vehicles. This part of methodology is implemented in the first section(Lane) of the jupyter notebook.

B. Behaviour of Vehicles (class Vehicle)

The Vehicle class defines the behaviour of vehicles on the motorway. In other words, it considers the various scenarios that can take place on a motorway like emergency braking, overtaking, crashing, etc. The class starts by initializing the initial state of the vehicles like their initial position, velocity, acceleration and rate of acceleration. It then performs Euler's integration to update the position, velocities and acceleration of the vehicles. This class also defines the overtaking maneuver of the vehicles in great detail. A vehicle will consider overtaking only if the vehicles in the front and the back are beyond the minimum threshold distance (defined in the constructor of the vehicle class) and also if there is no vehicle in the neighbouring lane. It also defines the recording process for the vehicles. An adjust velocity function is defined in this class to enable the vehicle to smoothly adjust their velocity and change lanes to avoid the crash, when another vehicle is closely approaching.

C. View of the Vehicles (class Surround)

The surround class enables a vehicle to have access to the next vehicles to the front, back, left and right side of the vehicle. Pictorial representation of the different elements in the surround class are shown in the figure 1. The *leftLane* and *rightLane* are references to next Lane and are None is there is no lane to the left or right. *left* and *right* are Boolean values that indicate that there is a vehicle in the critical region or not. Front (*leftFront, rightFront*) and back (*leftBack, rightBack*) are references to the Vehicle in the indicated region that is next to the current position. If there is no such vehicle, the references return None. The limits of the relevant regions are defined based on the current position +/- a number of car lengths +/- a distance that is a multiple of the current velocity, defined as time constants. The *Surround* class is the third section in the attached jupyter notebook.

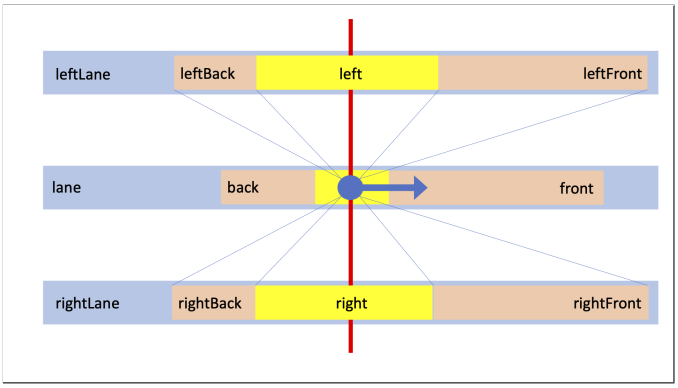

Fig. 1. Elements in the *Surround* class

D. Recording the Simulation (class Recorder)

The *Recorder* class is essentially responsible for recording all the different events that take place during the run time of the simulation. It defines the sub-processes for events like braking, crashing, changing lane and vehicle reaching the end of simulation and stores them into different columns of a pandas dataframe. The different possible events are listed below:

- **Change fast**: It is recorded when a vehicle in the slow lane overtakes another vehicle to move to a fast lane.
- **Done change slow**: It is recorder when a vehicle from a fast lane overtakes another vehicle to move to a slow lane.
- **Brake**: This event is recorded when a vehicle applies brakes at any point during the simulation run time (figure 3.
- **Crash**: it is recorded when there is not enough distance in the front or back or both of the vehicle and as a result, it crashes into another vehicle (figure 4).
- **End**: The end event is recorder when a vehicle reaches at the end of the simulation (figure 5).

```
def enter(self, vehicle, pos=0):      #defines the enter event for the vehicle
    self.vehicles.insert(0, vehicle)
    vehicle.pos = pos
    vehicle.lane = self
    vehicle.rec.record(vehicle, event="enter lane")

def leave(self, vehicle):             #defines the leave lane event for the vehicle
    vehicle.rec.record(vehicle, event="leave lane")
    vehicle.lane = None
    # in the meantime the vehicle may have have moved
    # to one of the next lane segments...
    lane = self
    while lane is not None:
        if vehicle in lane.vehicles:
            lane.vehicles.remove(vehicle)
            break
        else:
            lane = lane.next
```

Fig. 2. Recording of enter and leave lane events

IV. Simulation Models

Different simulation models were run after constructing various different classes as described in the previous section. For each simulation, a different set of parameters was chosen and different elements were introduced in each of them.

```
def emergencyBraking(self, v):   # defines the emergency braking procedure for vehicles

    def emergencyBrakingProcess(v):
        self.rec.record(self, 'brake')
        minΔt = 0.2
        self.dddx0 = (self.a_min-self.ddx0)/minΔt
        yield self.env.timeout(minΔt)

        self.updateOnly()
        self.dddx0=0
        self.ddx0=self.a_min
        v = min(v, self.dx0-2)
            # the brake time estimate is for perfect timing for
            # autonomous cars. For manual driving leave out the
            # -minΔt/2 or use a random element.
        Δt = max(0.5, (v-self.dx0)/self.ddx0 - minΔt/2)
        yield self.env.timeout(Δt)

        self.updateOnly()
        self.dddx0 = -self.ddx0/minΔt
        yield self.env.timeout(minΔt)

        self.updateOnly()
        self.ddx0 = 0
        self.dddx0 = 0


    ## The 'braking' bit prevents the interruption of an emergency breaking process
    self.braking = True
    self.processRef = self.env.process(emergencyBrakingProcess(v))
    try:
        yield self.processRef
    except simpy.Interrupt:
        pass
    self.processRef = None
    self.braking = False
```

Fig. 3. Emergency braking process in the *Vehicle* class

```
def crash(self, other):    # function for recording the crash event

    def recordCrash(self):
        self.rec.record(self, 'crash')
        self.running = False
        self.crashed = True
        self.dx0 = 0
        self.ddx0 = 0
        self.dddx0 = 0

    if self.running:
        print(f"Crash p{self.id:d} into p{other.id:d} at t={self.t0:7.3f} x={self.x0:7.1f}")
        recordCrash(self)
        if other.running:
            recordCrash(other)
```

Fig. 4. Crash event defined in the *Vehicle* class

```
def updateOnly(self):
    if self.crashed:
        return False
    t = self.env.now
    if t < self.t0:
        return False
    if self.running and t > self.t0:     # Euler integration for finding the position, velocity and accelaration
        dt = t - self.t0
        ddx = self.ddx0 + self.dddx0*dt
        dx = round(self.dx0 + self.ddx0*dt + self.dddx0*dt*dt/2,4)
        Δx = self.dx0*dt + self.ddx0*dt*dt/2 + self.dddx0*dt*dt*dt/6
        x = round(self.x0 + Δx, 2)
        self.t0, self.x0, self.dx0, self.ddx0 = t, x, dx, ddx  #Result of Euler integration returns time, position,
                                                                # velocity and acceleration
        self.pos = round(self.pos+dx, 2)
        # update lane information if necessary
        if self.pos >= self.lane.length:
            nextPos = self.pos - self.lane.length
            nextLane = self.lane.next
            self.lane.leave(self)
            if nextLane is None:
                self.rec.record(self, event='end')    #Record the end event when the nextLane is none i.e when there is
                self.running = False                  # no lane ahead
                return False
            else:
                nextLane.enter(self, pos=nextPos)
                if self.oldLane is not None:
                    self.oldLane = self.oldLane.next
    return True
```

Fig. 5. End event defined in the vehicles class

1) **Model for Traffic Generation:** We calculate the following four quantities for each of the simulations in order to get insights into how the different parameters are affecting the results of our simulation.
   - *Average Travelling Time*: It is the average time taken by a vehicle to cross the entire motorway. In our case, it is calculated by taking the difference between the 'end' and the 'enter lane' events for each vehicle that has made it through the end of the simulation and then averaging these times for the number of vehicles chosen.
   - *Net Throughput*: Throughput can be defined as the Cumulative volume across all the lanes going in one direction measured and averaged over a prolonged period of time. In the context of our study, it is calculated by taking the difference between the last vehicle exiting the motorway and the first vehicle exiting the motorway.
   - *Average Speed*: Average speed of the vehicle is the ratio of the total length of the lane and the average travelling time for the motorway
   - *Average Density of Traffic*: The density of traffic can be defined as the cumulative volume of the traffic per unit average speed. In other words, it can be calculated by taking the ratio of the net throughput and the average speed.

In addition, there are some global constants which are passed as the parameters to the constructor inside the *Vehicle* class. These are listed below:

- *CRITICAL_TIME_TOLERANCE*: Can be defined as the time when at current speed difference a crash might occur within that number of seconds. It is chosen as 4 seconds.
- *LANE_CHANGE_TIME*: Maximum time it takes for a vehicle to change the lane. It is chosen as 3 seconds but it's value really depends on the type of vehicle and also somewhat on the behaviour of the driver.
- *MIN_TIME_DIFF*: Minimum difference in times to trigger overtaking. it is taken as 1 second.
- *MIN_SPEED_DIFF*: Minimum speed difference to trigger overtaking process. It is chosen as 2 m/s.
- *CAR_LENGTH*: Typical length of a car. It's value is taken as 4 metres.
- *FAR_AWAY_IN_FRONT*: Distance at which a car in front can be ignored. It's value is also dependent on various factors, but to be on the safe side, we have taken it to be 200 meters.
- *FAR_AWAY_IN_BACK*: Distance at which a car behind can be ignored. It's value is taken as 80 metres.

The simulation study can be divided into two parts.The first task involves simulating a two lane motorway and then calculating the above mentioned quantities. The second task involves incorporation of *freemotorway speed()* function to calculate the velocities of the

vehicles. Each task encapsulates some aspects of the modelling. For accomplishing this task, we have used the code provided in the class and added the following components to it:

2) **Model for Behaviour of Human Drivers**: To simulate the behaviour of human drivers, the following components have been added:
   - *chooseLane() method*: This function allows a vehicle to choose between the two lanes at random but with specific weights given to each lane. For the *left/slow lane*, we have given a weight of 0.80 since majority of the vehicles will be starting from this lane only. The *right/fast lane* is given a weight of 0.20 since only a small fraction of incoming vehicles pass through this lane.
   - We have used *different inter-arrival times* for each of the two lanes *IAT_fast* and *IAT_slow*. We ran different simulations with different values for the two inter-arrival times.
   - A condition is added to the main simulation loop allowing vehicles starting from the left lane to use *IAT_fast* and those starting from the right lane to use *IAT_slow*.
   - For calculating the *iat*, a *uniform distribution* has been used instead of an *expovariate distribution* to reduce the variation. Moreover, expovariate distribution generated a few extremely small values like 0.0048 for the inter-arrival time which caused crashes. the simulation results with expovariate distribution have also been reported.
   - The second task is an extension of the first task to simulate the human driving behaviour. In this task, we will be using the *freeMotorwaySpeed()* function from the traffic data generation file for generating the speed of the vehicles.
3) **Model for Vehicles**: We also incorporated the behaviour of different types of vehicles like electrical and diesel. We choose different values of minimum and the maximum acceleration for each of the type. The maximum speeds used in the motorway speed function is also changed according to the vehicle type. Although, the electrical cars can attain less maximum velocities than diesel cars, they can accelerate much faster [13].

We have ran various simulations but below are the results of the simulations that gave the optimum results i.e maximum throughout and optimal average travelling time. The lane length is 3 Km or 3000 metres for all the simulations

A. Simulation 1

In this simulation, we use single inter-arrival time with the random speed variation of vehicles.The parameters are listed below and the position-time graph is shown in the figure 6.

- Number of cars(N): 3000
- Inter-arrival time(IAT): 5 sec
- Distribution used for *iat*: *random.uniform()*; Range: IAT/10 to IAT+10

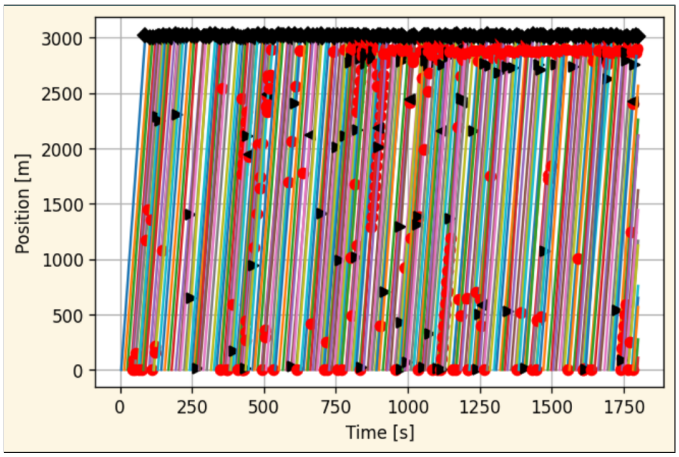

Fig. 6. Position-time plot for the first simulation with single inter-arrival time for both lanes

Interpretation: Looking at the position time graph from figure 6, we can conclude that there are very few crash events in the simulation. This can be attributed to the fact that there is much less variation in the inter-arrival time due to the use of uniform distribution. The inter-arrival time is also low which allows more vehicles to reach at the end of the simulation which allows for maximum throughput.

B. Simulation 2

For this simulation, we use the *expovariate()* distribution for calculating the inter-arrival time (IAT). And we choose the single value for IAT for both the lanes. The parameters for the simulation are given below and the position-time graph is depicted in the figure 7

- Number of cars(N): 2000
- Inter-arrival time(IAT): 18 sec
- Distribution used for *iat*: *random.expovariate(1.0/IAT)*

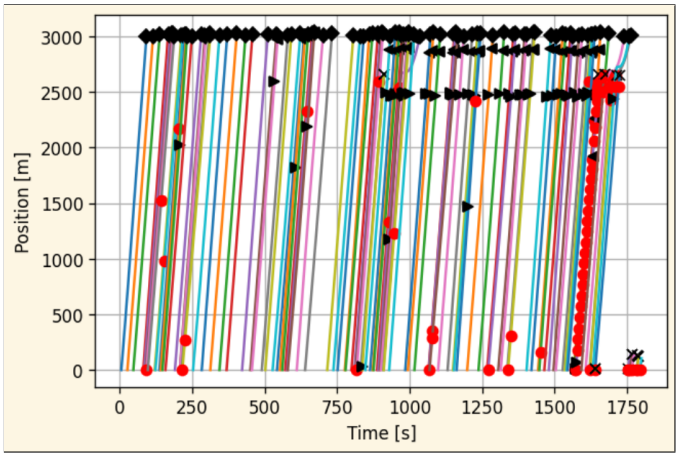

Fig. 7. Position-time plot for the second simulation with single inter-arrival time (calculated using expovariate distribution) for both lanes

Interpretation: We can see from the position time plot that there are a few crashes towards the end of the simulation

(> 1600 seconds). The throughput is significantly reduced as compared to the simulation 2. This is because of the rapid variations in the inter-arrival time because of the expovariate function. The minimum value for the expovariate function goes as low as 0.004 which in turn causes vehicles to crash since there is not enough time for overtaking. Using expovariate distribution, the throughput came out to be 203 cars/hours and the average travelling time of about 92 seconds. The average speed came out to be around 32.73 m/s.

C. Simulation 3

For this simulation, we use the *chooseLane()* method that allows a vehicle to choose randomly between the two lanes. We also use different inter arrival times for both lanes and the *freemotorwaySpeed()* function to generate the speed profiles of the vehicles. the parameters of the simulations are listed below and the position time plot is shown in figure 8.

- Number of cars(N): 2000
- Inter-arrival time(IAT): *IAT_fast:* 13 seconds, *IAT_slow:* 13 seconds.
- Distribution used for *iat*: *random.uniform()*; Range: IAT/10 to IAT+10 (for both inter-arrival times)

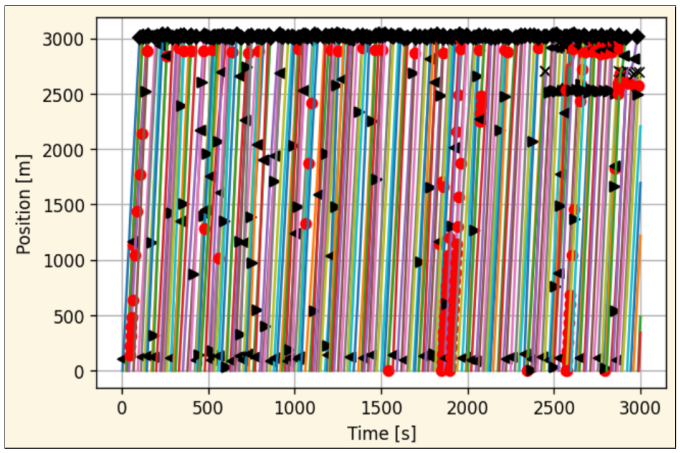

Fig. 8. Position-time plot for the third simulation with different inter-arrival time for both lanes

Interpretation: As we can clearly see form the position time plot that there are very few crashes and that too towards the end of the simulation. The parameters chosen produce the throughput value of 224 cars/hours, an average travelling time of 97.71 m/s and an average speed of 33.2 m/s. The comparatively longer average travelling time can be attributed to the relatively high values for *IAT_fast* and *IAT_slow*.

D. Simulation 4

For this simulation, we introduced the different types of vehicles. the two types of vehicles chosen are *diesel* and *electrical*. The simulation chooses the type of vehicle at random. For this simulation, we choose the single inter-arrival time for both the lanes calculated through a uniform distribution as in simulation 3. The parameters used for the simulation are listed below and the position-time graph is shown in the figure 9.

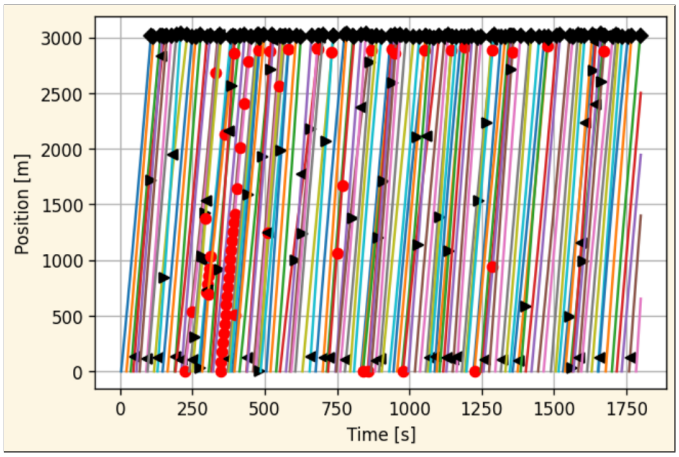

Fig. 9. Position-time plot for simulation 4 with single inter-arrival time calculated through uniform distribution and the random choice for the type of vehicle between *electrical* and *diesel*

- Number of cars(N): 2000
- Inter-arrival time(IAT): 5 seconds.
- Distribution used for *iat*: *random.uniform()*; Range: IAT/10 to IAT+10 (for both inter-arrival times)

Interpretation: As can be seen from the position-time graph that the simulation has no crashes in the given run time. The weights were assigned for the random choice of vehicles types between *diesel* and *electrical*. The maximum and minimum acceleration for both the types was passed in the arguments of the vehicle class. These values are listed in figure . The throughput value obtained was around 262 cars/hours while the average travelling time and speeds were 97.77 seconds and 30.68 m/s. The drastic change in the average velocity is because of the fact that the electrical vehicle, although attain less maximum velocity than diesel cars, they can accelerate and decelerate more quickly than diesel cars which means they can move at a greater velocity on the motorway and still suffers less crashes than the diesel cars.

The results obtained in the simulations have been summarized in figure 10

V. Conclusion and Future Works

Traffic congestion is one of the major problems in the world today. Simulating various aspects of the real world scenarios allows us to then introduce the required changes in the real situations. In this study, our aim was to simulate the behaviour of different types of vehicles on a motorway. Many different simulations were run with different combination of parameters and models for behaviour of human drivers. The ones giving the maximum throughput were reported. As we can see from the previous section, the simulation in which a single value for the inter-arrival time was used with random speed distribution attains the maximum throughput of around 500 cars/hour. The simulation in which different inter-arrival times were chosen gave the net throughput of about 224 cars/hour and an average

Simulation	Simulation 1	Simulation 2	Simulation 3	Simulation 4
Aspect	Single IAT with random speeds	Single IAT with random speeds, randomSpeedvaritaion() function for generating velocities	Diferent IATs for both lanes, chooseLane() method for randomly choosing lane (slow/fast), freemotorwaySpeed() function for generating velocities	Single IAT, chooseVehicle() method for randomly choosing vehicle between electrical and diesel, freemotorwayspeed() function for generating velocities
N	3000	2000	2000	2000
IAT	5 seconds	18 seconds	IAT_fast = 13 seconds, IAT_slow = 18	5 seconds
Distribution used for calculating IAT	random uniform distribution	expovariate distribution	random uniform distribution	random uniform distribution
Net_Throughput (cars/hours)	485.74	202.54	223.97	261.85
Average Travelling Time (seconds)	90.35	91.65	97.71	97.77
Average Speed (m/s)	33.2	32.73	30.7	30.68
Traffic Density (cars/Km)	1.87	1.71	2.04	2.18

Fig. 10. Summary of the results obtained from simulations

travelling time of about 98 seconds. These values closely represent the real world cases of a multi-lane motorway.

The future works can consider various other aspects of the driving behaviour like reaction times, lane change maneuvering behaviour. The lane merging behaviour can also be introduced in the simulation classes which will drive the vehicles to adjust their velocities and adjust their position in two lanes from three lanes. The introduction of autonomous cars will also help in the reduction of the traffic congestion.

REFERENCES

[1] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," Tech. Rep., 2015.

[2] T. Holstein, G. Dodig-Crnkovic, and P. Pelliccione, "Real-world ethics for self-driving cars," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, 2020, pp. 328–329.

[3] N. Hara and D. Marinescu, "Self-organising motorway trafic management," 2019, [Accessed 4th May 2021]. [Online]. Available: https://www.tcd.ie/futurecities/research/mobility/traffic-management.php

[4] A. Khan, F. Ullah, Z. Kaleem, S. U. Rahman, H. Anwar, and Y.-Z. Cho, "Evp-stc: Emergency vehicle priority and self-organising traffic control at intersections using internet-of-things platform," *IEEE Access*, vol. 6, pp. 68 242–68 254, 2018.

[5] N. N. N. Azlan and M. M. Rohani, "Overview of application of traffic simulation model," in *MATEC Web of Conferences*, vol. 150. EDP Sciences, 2018, p. 03006.

[6] P. G. Gipps, "A model for the structure of lane-changing decisions," *Transportation Research Part B: Methodological*, vol. 20, no. 5, pp. 403–414, 1986.

[7] S. Srikanth, A. Mehar, and K. G. N. V. Praveen, "Simulation of traffic flow to analyze lane changes on multi-lane highways under non-lane discipline," *Periodica Polytechnica Transportation Engineering*, vol. 48, no. 2, pp. 109–116, 2020.

[8] H. Hu, Z. Lu, Q. Wang, and C. Zheng, "End-to-end automated lane-change maneuvering considering driving style using a deep deterministic policy gradient algorithm," *Sensors*, vol. 20, no. 18, p. 5443, 2020.

[9] E. Rodaro and Ö. Yeldan, "A multi-lane traffic simulation model via continuous cellular automata," *arXiv preprint arXiv:1302.0488*, 2013.

[10] R. Harb, E. Radwan, and V. V. Dixit, "Comparing three lane merging schemes for short-term work zones: A simulation study," *International Scholarly Research Notices*, vol. 2012, 2012.

[11] S. Labib, H. Mohiuddin, I. M. A. Hasib, S. H. Sabuj, and S. Hira, "Integrating data mining and microsimulation modelling to reduce traffic congestion: A case study of signalized intersections in dhaka, bangladesh," *Urban Science*, vol. 3, no. 2, p. 41, 2019.

[12] D. Sulejic, R. Jiang, N. R. Sabar, and E. Chung, "Optimization of lane-changing distribution for a motorway weaving segment," *Transportation Research Procedia*, vol. 21, pp. 227–239, 2017.

[13] F. Brito, J. Martins, D. D. R. Pedrosa, V. D. F. Monteiro, and J. L. Afonso, "Real-life comparison between diesel and electric car energy consumption," 2013.