Simulating Traffic Flow within a Closed Loop

K. J. Dworski

University of Southampton

Abstract

This paper intends to explore the effect of various factors that may influence the passage of traffic in a (near) circular loop. The system contains an amount of cars, n, within a ‘loop’ with a radius, r. The system is simulated to run continuously given time T and is modeled using an approximation of the Intelligent Driver Model (IDM). In general, a system is only as fast as it’s slowest member; and traffic ‘waves can propagate through the model, given certain external factors that may influence any particular vehicle within the model. Inspiration is given to 2 models that I found available online (Volkhin, n.d.) (Treiber, n.d.)

Keywords: Intelligent Driver Model, Traffic Flow

Simulating Traffic Flow within a Closed Loop

## Introduction

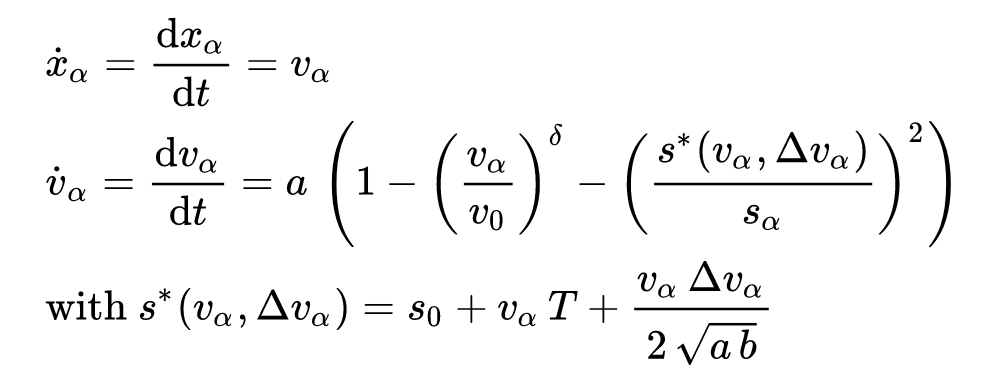
Traffic and travel are 2 things that almost everyone who has even wanted to transport themselves from point A to point B is aware of. Initially, modelling traffic seems to be a simple ordeal, but modelling traffic with some degree of accuracy can be quite troubling. There are many different methods, techniques, and simplifications that need to be considered, as well as the type of model that is being used, and the intended purpose. Simulating traffic flow is, of course, also much safer, and more practical than using the real world – and the latter isn’t always an affordable, or practical, option. Ideas can be tested, and data can be collected to help optimize actual traffic systems in the real world.

The Intelligent Driver model is a time-continuous car following model for the simulation of traffic. It treats each vehicle as a dynamic system, where each vehicle reacts to the car it is following. In this paper, I describe the model I used, the equations behind the motion of each vehicle, and look at the model itself and analyse the data.

## Method, Assumptions and Simulation Analysis

The simulation treats the road as a one-dimensional loop, along which vehicles travel at varying speeds. The vehicles are treated as if they are following one, long continuous track, and is only allowed to travel forwards behind its predecessor. For the purposes of the first analysis, all vehicles are assumed to travel with uniform acceleration, deceleration, and maximum velocity. The use of pygame is especially useful, as it self-optimizes for multithreading and memory usage, optimising the program quite significantly. The only variance between vehicles are their length, which is randomly determined, and their colour, also randomly determined; the length only affects the separation between vehicles, the colour is purely cosmetic.

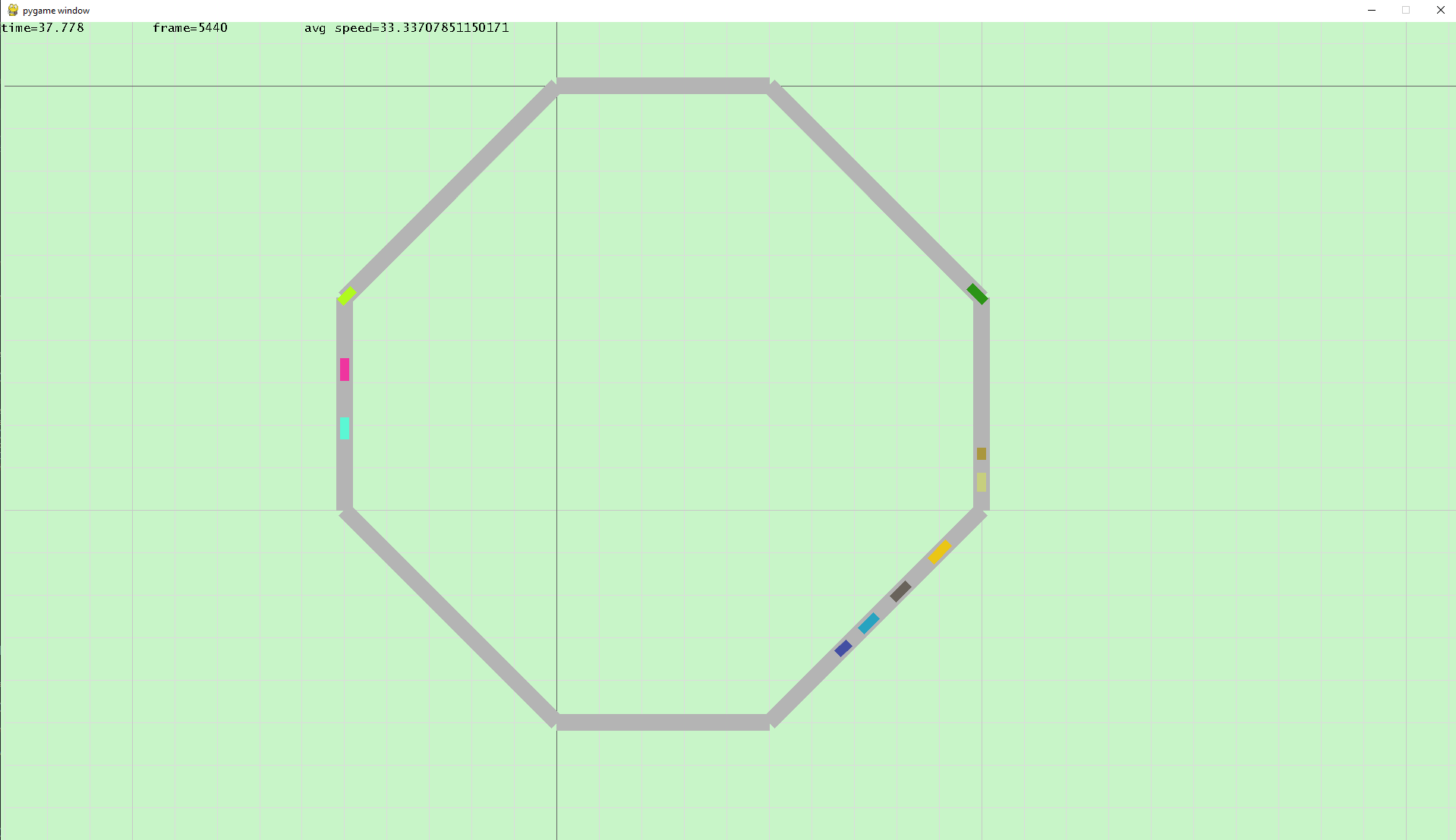
As discussed previously, I chose to use the Intelligent Driver Model as my inspiration for this simulation, as it is the most up to date and capable model for modelling a vehicle’s velocity with regards to realistic vehicle properties. The model treats the dynamics of a vehicle as a set of the following equations:



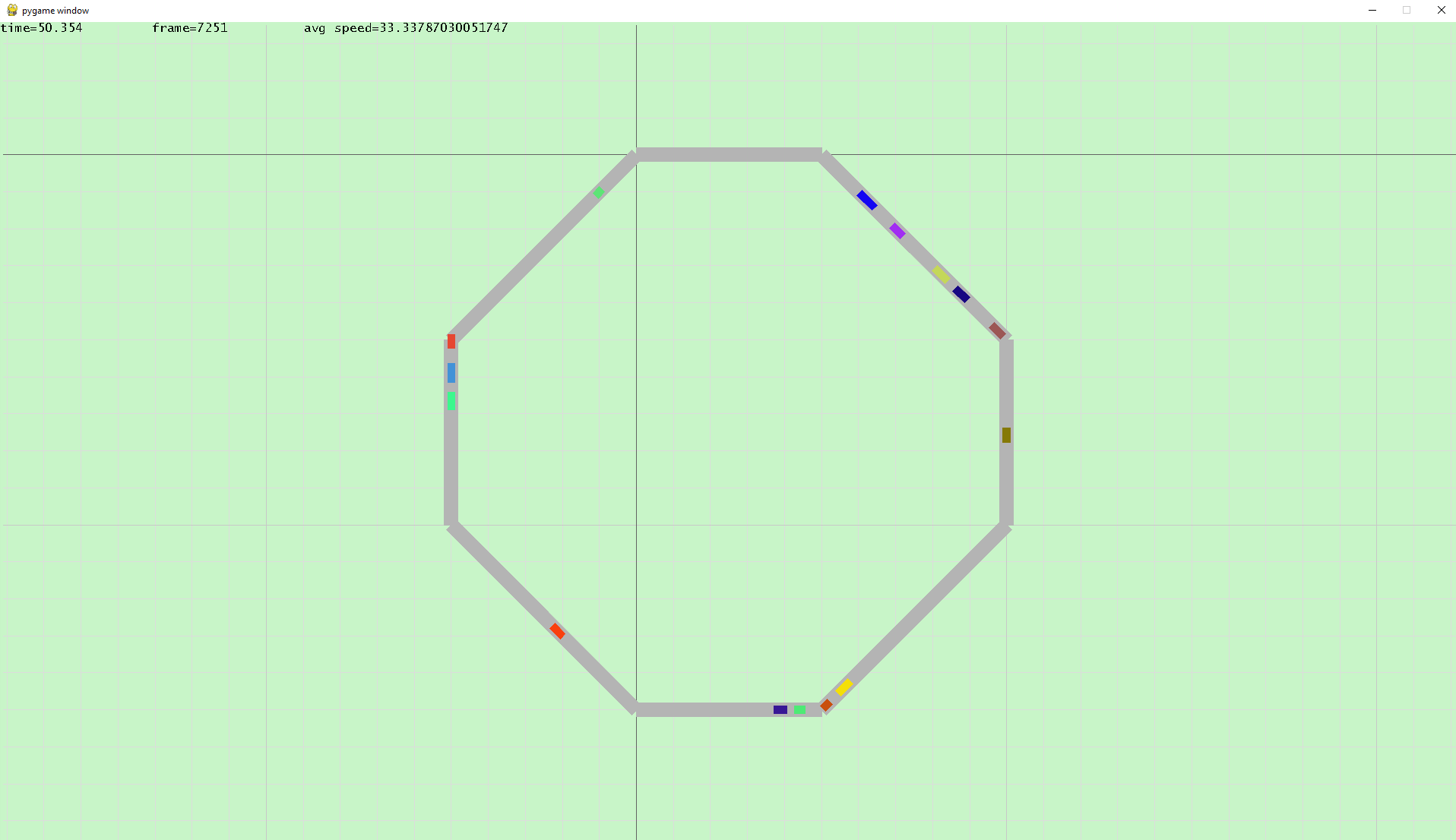
For a vehicle alpha, it’s velocity is treated as the derivative of its distance (as per Newtonian mechanics). The acceleration is treated as a free term, denoted above by the portion to the power of delta, and the interaction term, denoted above by the portion to the power of 2. The free term can be described as the vehicles ideal ‘desire’ to accelerate to the ideal velocity V0, and it would do so given a free, open road. The interaction term denotes the interaction with a vehicle ahead of it, such that the vehicle comfortably avoids a collision, and will come to a complete stop at a distance s0 behind the vehicle in front of it. Also, in the above, a and b are the vehicle’s comfortable acceleration and decelerations respectively and T is the minimum time to the vehicle in front. Delta is the acceleration exponent, and for the purposes of this simulation is taken to be 2.

For the purposes of the simulation, several assumptions had to be made. For the first, analysis of the time for traffic to settle to an equilibrium on a loop, all vehicles are treated with the same velocity and acceleration parameters. All the vehicles are also treated to start from a standstill, in a sort of similar way to how cars would drive one by one from a driveway onto a main road. One thing that added an element of randomness, was that each vehicle would occasionally decide to slow down, simulating an unexpected event, say someone crosses the road, or a wild animal appears.

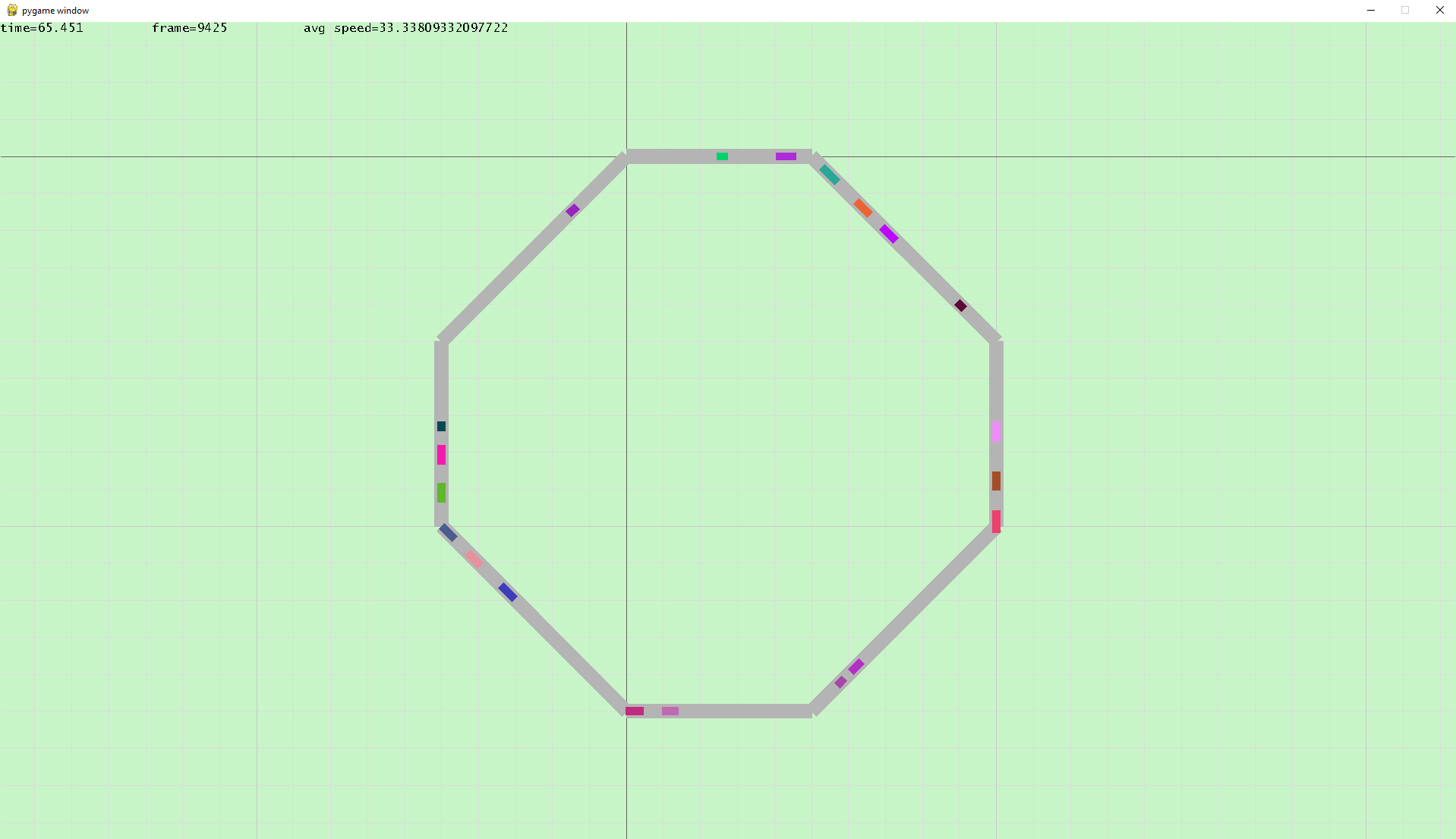
Running the program for various amounts of vehicles yields interesting results; and occasionally pygame would have a hiccup due to personal programming limits. Below are figures displaying some of the results of running said simulations, and the numbers involved with these. The maximum speed for each of these simulations was set to 33.333 m/s, roughly equivalent to 120 km/h, and any difference between this and the displayed maximum speed is due to Pygame and Python limitations:



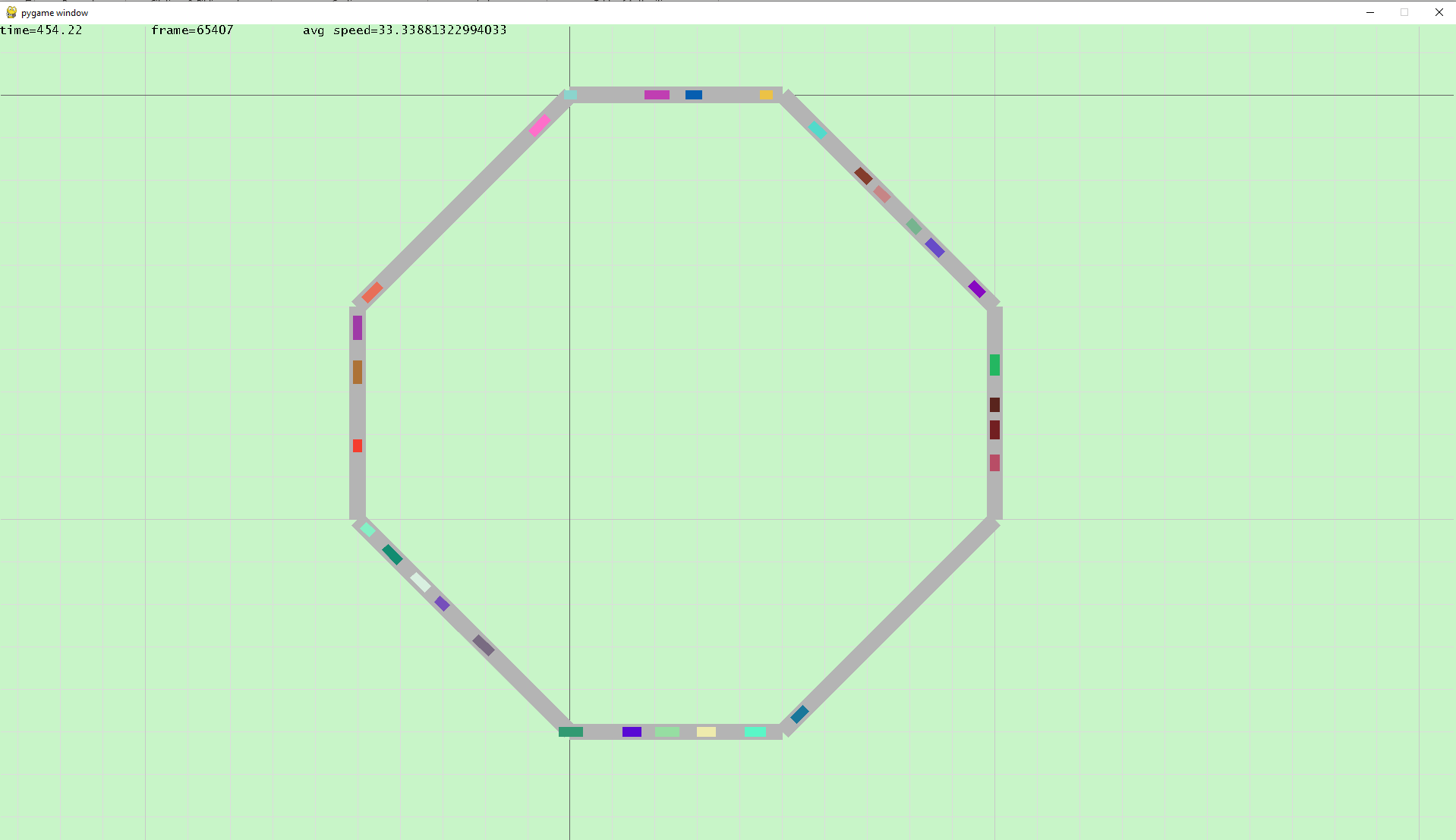
**Figure 1**: This figure shows the resultant flow of one of the simulations when running with only 10 vehicles. Time taken for this simulation to reach equilibrium was 37 seconds.



**Figure 2**: This figure shows the resultant flow of one of the simulations when running with only 15 vehicles. Time taken for this simulation to reach equilibrium was 50 seconds.

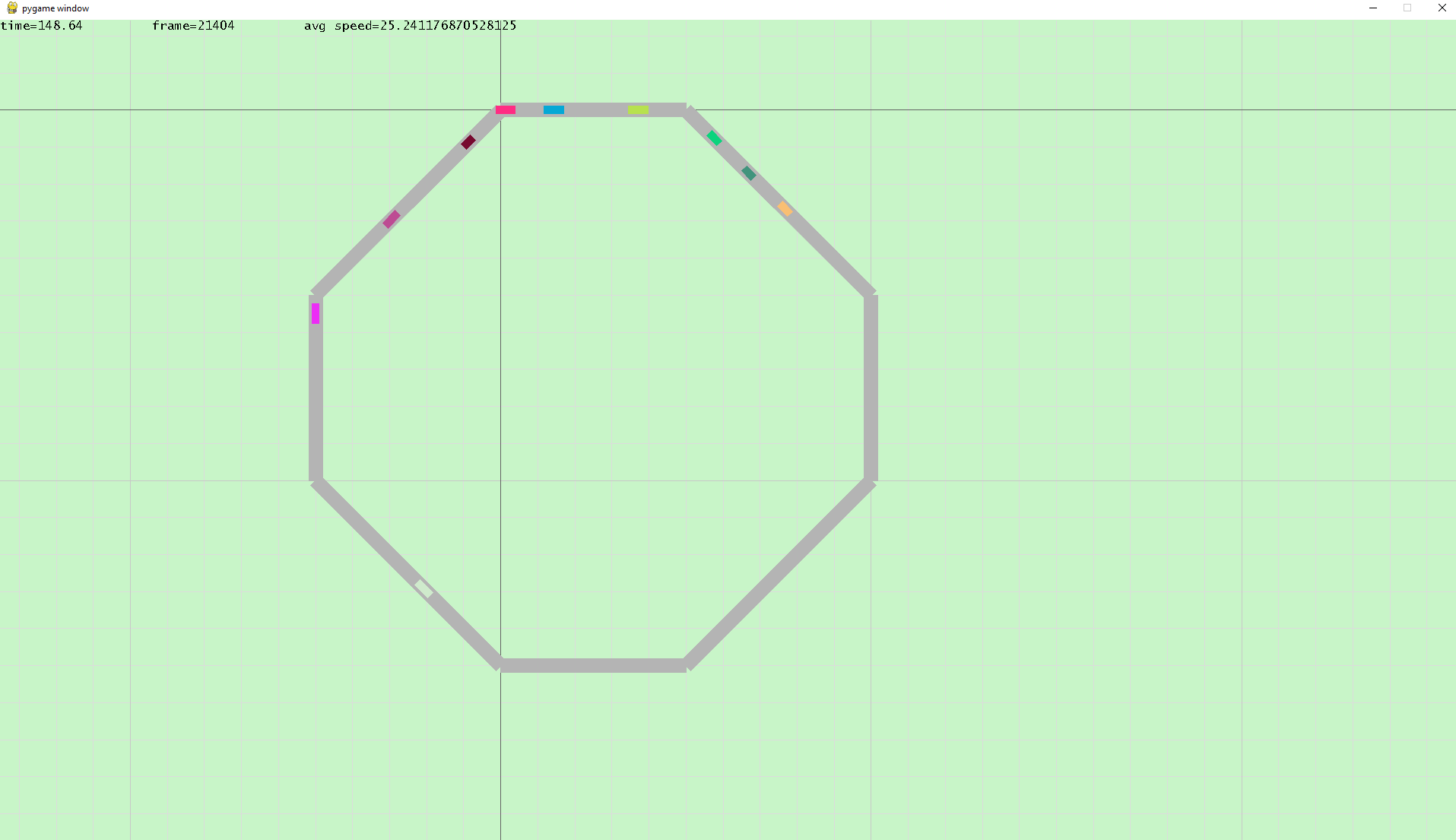


**Figure 3**: This figure shows the resultant flow of one of the simulations when running with only 20 vehicles. Time taken for this simulation to reach equilibrium was 65 seconds.

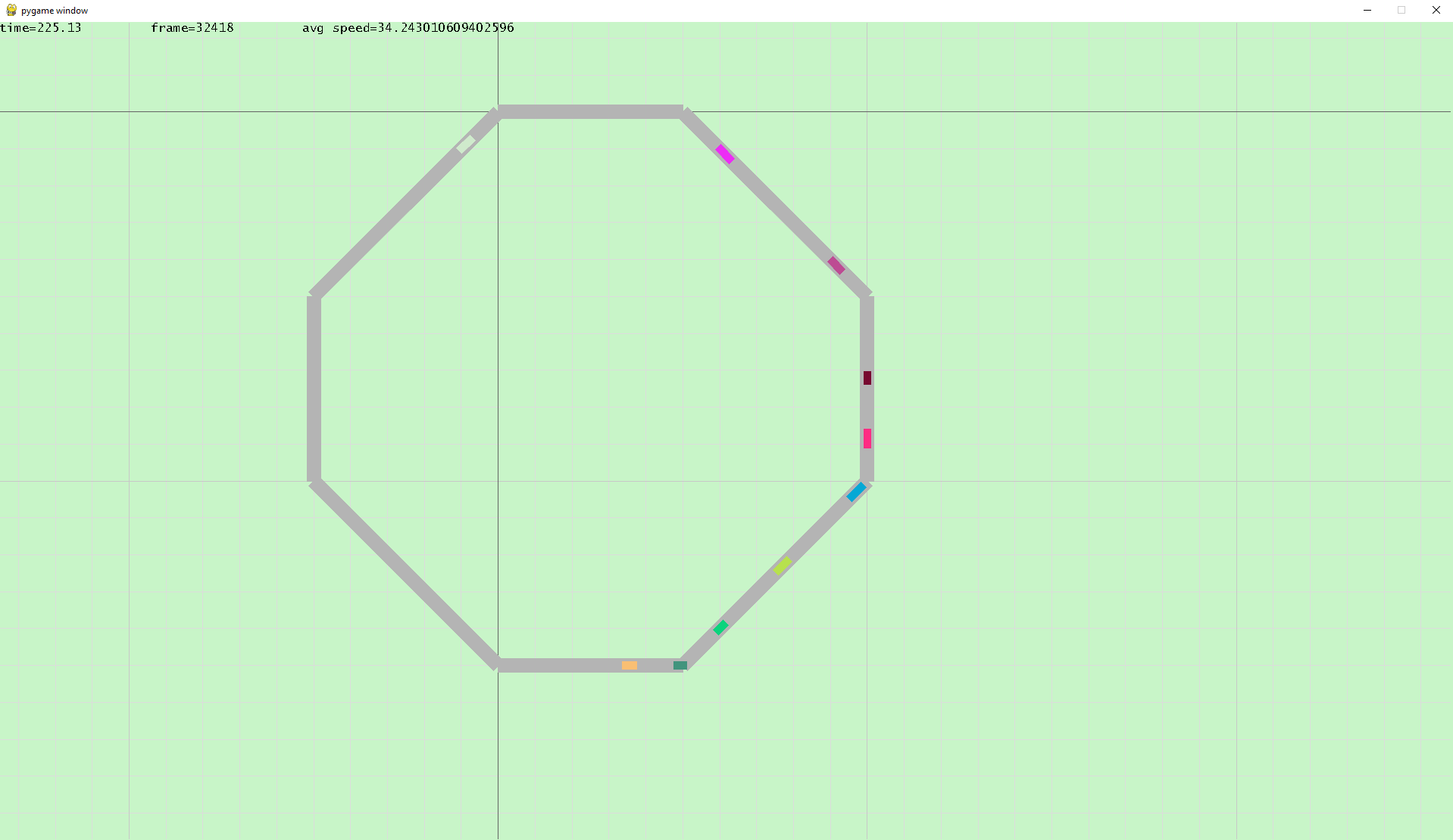


**Figure 4**: This figure shows the resultant flow of one of the simulations when running with only 30 vehicles. Time taken for this simulation was significantly longer than previous simulations, as the time to reach equilibrium was 454 seconds.

Next, the following simulations were taken with vehicle max speeds randomly assigned between 2 numbers. All other vehicle properties were kept the same as previously. For ease of keeping the simulation consistent and quick to figure out results or trends, the vehicle numbers were kept at 10 each time.



**Figure 5**: This figure shows the results when running 10 cars at max speeds varying from 20 to 33 m/s. The average final speed achieved of the simulation was roughly equal to the maximum velocity of the slowest car, with some variation due to the cars behind decelerating and accelerating.



**Figure 6**: This figure shows the results when running 10 cars at faster max speeds, varying from 33 to 75 m/s. The average final speed achieved of the simulation was roughly equal again to the maximum random velocity of the slowest car, with again some variation due to the cars behind decelerating and accelerating.

It is interesting to note the grouping that occurs from when one vehicle would slow down to react to an event, and then when the others would bunch up behind it. This is visibly shown in figures 1 to 4, where the vehicles max velocities and accelerations were uniform, and the vehicles would all still bunch up from a braking event that a car may undergo, and these bunching patterns are still visible when the vehicles all reach their max velocity. For example, figure 1 shows two separate, distinct grouping patterns on both the left and the right-hand side of the loop. Figure 4 also shows some clear remnants of grouping, with larger gaps between groups of vehicles, although the vehicles are much more uniform around the loop.

The grouping is much easier to see when looking at the results in figures 5 and 6. The vehicles also liked to keep a larger spacing when going at slower velocities collectively.

We can make a table of the results from the first experiment, and then likewise plot them using matplotlib. Please note, all results are approximations due to the highly approximate nature of the simulation, as well as the unpredictability:

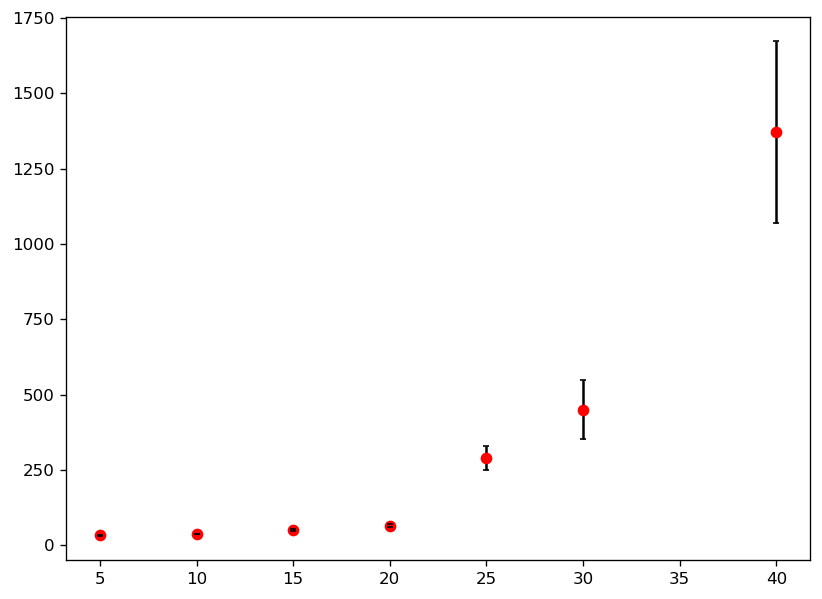
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Number of Vehicles | Time to Equilibrium 1 | Time to Equilibrium 2 | Time to Equilibrium 3 | Mean | Variance |
| 5 | 33 | 33 | 34 | 33.333 | 0.3333 |
| 10 | 37 | 38 | 38 | 37.666 | 0.3333 |
| 15 | 50 | 52 | 53 | 51.666 | 2.3333 |
| 20 | 65 | 69 | 71 | 68.333 | 9.3333 |
| 25 | 280 | 291 | 311 | 294 | 247 |
| 30 | 454 | 510 | 466 | 476.66 | 869.33 |
| 40 | 1371 | 1355 | 1488 | 1404.7 | 5272.3 |

**Table 1**: A table showing the results of varying vehicle amounts multiple times, to work out the rough mean and variance of the data from the simulation

|  |  |  |
| --- | --- | --- |
| Slowest Vehicle Vmax | Fastest Vehicle Vmax | Average Speed of Vehicles |
| 25.2131308 | 31.1731805 | 25.2411768 |
| 34.1411916 | 71.9910137 | 34.2430106 |
| 72.9391103 | 101.3134404 | 73.2199434 |

**Table 2**: A table showing the results of varying the vehicle max velocities randomly, and the average vehicle speed of the simulation after an arbitrarily long amount of time (Note: The system never seemed to reach complete stability, as there would always be some fluctuation, the simulation was running until t reached 7500 seconds)

Plotting table 1 in matplotlib, with errors, gives us the following graph:



**Figure 7**: A graph plotted to Table 1, visualizing the results of running the simulation with various amounts of cars, and timing how long (vaguely) it takes to reach the equilibrium point where all vehicles are travelling, uninterrupted, without slowing down

We can see that, given a vehicle loop of vehicles, if each of them has a random chance of slowing down suddenly, the propagation of ‘slowing down’ through the vehicle population causes the simulation to take (seemingly) exponentially longer as the number of vehicles in the simulation increases. This remained the case, regardless of what the vehicle max speed was set to, the only difference being the eventual velocity that every vehicle would travel at once the simulation reached the equilibrium point. Somewhat not visible to show, but what is coded within the program, is the ability to slow down a random vehicle within the simulation by right clicking – doing so shows how traffic seems to bunch up around a loop, and how the flow of this phantom traffic goes against the flow of traffic. This effect can be easily exaggerated by modifying the acceleration rate to a very slow rate, such as around 0.5 m/s^2.

Results from Figure 7 would also suggest that the solution to traffic isn’t as simple as just increasing the amount of vehicles, as this just creates more congestion on finite space and causes the entire system to become slower, and would suggest

Table 2 shows that, regardless of the speed of the remaining vehicles in the circuit, if no alternative option is available, the network is only as fast as it’s slowest component, and that is the limiting factor. A case could from these results, therefore, be made to provide alternative transport links between 2 points or force every vehicle to go at a faster rate, as simply speeding yourself doesn’t cause a net increase in the speed of the simulation, and in a real context only causes more danger than good. In fact, from the simulations, certain vehicles going exceedingly fast seemed only to cause more traffic, as vehicles that speeded tended to brake and decelerate much harder than their counterparts, causing vehicles following behind to likewise need to overcompensate and brake harder, causing the system to jam and a phantom traffic jam to occur.

## Conclusions

A restricted vehicle loop, with a vehicle population N, was tested to simulate traffic flow in a circuit, for the case of the Intelligent Driver Model and the equations associated with it. From the data, we analyzed that more vehicles within the system just exponentially increases the time taken for the vehicles to reach their maximum velocity, thus prolonging travel times and lowering network efficiency. We can, therefore conclude, that a more optimum solution towards general traffic efficiency in a system, such as a road network, or a closed loop, would be to have larger vehicles carrying more passengers at once, as opposed to sticking more vehicles onto the network; or to provide alternate routes between locations, and to always stick to the speed limit as doing otherwise can actually cause more traffic than if everyone travels uniformly.

References

Cervero, R. (1984). Journal Report: Light Rail Transit And. *Journal of the American Planning Association*, 133-147.

Johan Olstam, V. B. (2019). Modelling Eco-Driving Support System for Microscopic Traffic Simulation. *ournal of Advanced Transportation, vol. 2019*, 16.

Martin Treiber, A. H. (2000). Congested Traffic States in Empirical Observations and Microscopic Simulations. *Physical Review E 62, 1805-1824*.

Treiber, M. (n.d.). *Traffic Simulation DE*. Retrieved from https://traffic-simulation.de/ring.html

Volkhin. (n.d.). *Volkhin Road Traffic Simulator*. Retrieved from http://volkhin.com/RoadTrafficSimulator/

Appendix: Techniques and Methods

The program was coded using, primarily, the pygame library in Python. This was installed to the Spyder IDE and runs the simulation as a continuous loop. Classes were used to help simplify the coding and debugging of the program. The IDM that I based my program and its equations from was developed by Treiber, Hennecke and Helbing in 2000 (Martin Treiber, 2000), who used it to analyse congestion forming from intersections, gradients, and any other inhomogeneities; and it is a way of modelling car-following microscopic traffic. Microscopic traffic flow is used all over academia and wider fields, from the (obvious) simulating traffic flow in a closed system to determine more optimal road routes or designs, to designing an optimal light rail transport system (Cervero, 1984), to investigating the environmental effects of more environmentally friendly driving methods (Johan Olstam, 2019).

The use of classes was particularly new to me, and I partially decided to use them to learn and challenge myself, but also the organisational element of using classes, and functions, and the use of self properties were particularly helpful. A very similar method of traffic management, as I discovered from emailing to ask, was used in the videogame Cities: Skylines; although the full methods used there are much more complex due to varying vehicle types, masses, and properties. The use of pygame seemed to me to be the easier, visual approach, as it very easily draws out the vehicles and the track they follow with ease, and it’s relatively easy to follow, although very temperamental. While this may not have been the best choice, it gives an easy visual to use and analyse, and the Intelligent Driver Model used is also a simple to follow, yet incredibly accurate and realistic, method.