Contents

List of Figures

List of Tables

1 Introduction

1.1 Modeling and Simulation Process

In general, Modeling and Simulation process is defined as a framework for activities, actions, and tasks required to develop a simulation. The process includes project description, conceptual model, simulation model, and so forth shown in Figure 1.

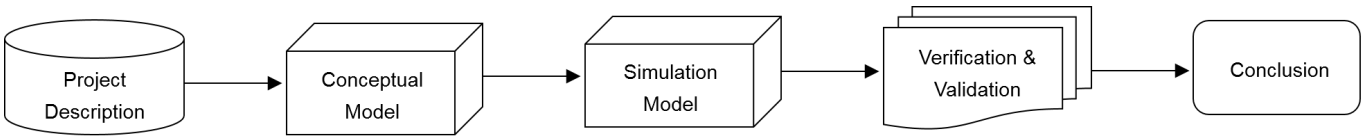

Figure 1: General Modeling and Simulation Process

In this project, we have implemented the Modeling and Simulation process into the traffic flow model with the real data called as NGSIM. The following chapters will describe how each process has been completed in the specific example, namely Atlanta Peachtree Street Traffic simulation.

1.2 Project Management

To systematically and efficiently complete this project, we implemented a very structured program management approach, which is shown in Figure 2. In addition, in order for us to facilitate good communications, we created a group chat environment throughout Facebook messenger and sharing the progress on the project. For this project, Wendi Ren was in charge of completing Cellular Automata simulation model. Yuanlai Zhou had the responsibility to finish Event-oriented simulation model. Lastly, Junghyun Kim had been working on Process-oriented simulation model.

Tasks	3/11 ~ 3/17		3/18 ~ 3/24		3/25 ~ 3/31		4/1 ~ 4/7		4/8 ~ 4/14		4/15 ~ 4/23	
Kick-off meeting												
Establish problem statement												
Study conceptual model												
Analyze input datasets												
Investigate simulation models												
Create Python codes for simulation models												
Verification and Validation												
Conduct Design of Experiment												
Analyze output results												
Trouble shoot as required												
Write report												
Turn in the report												

Figure 2: GANNT chart for this project

2 Project Description

In this project, we are expected to simulate the traffic flow at Peachtree Street in Atlanta to understand simulation methodology and software development. The street used for this project is viewed in Figure 3.

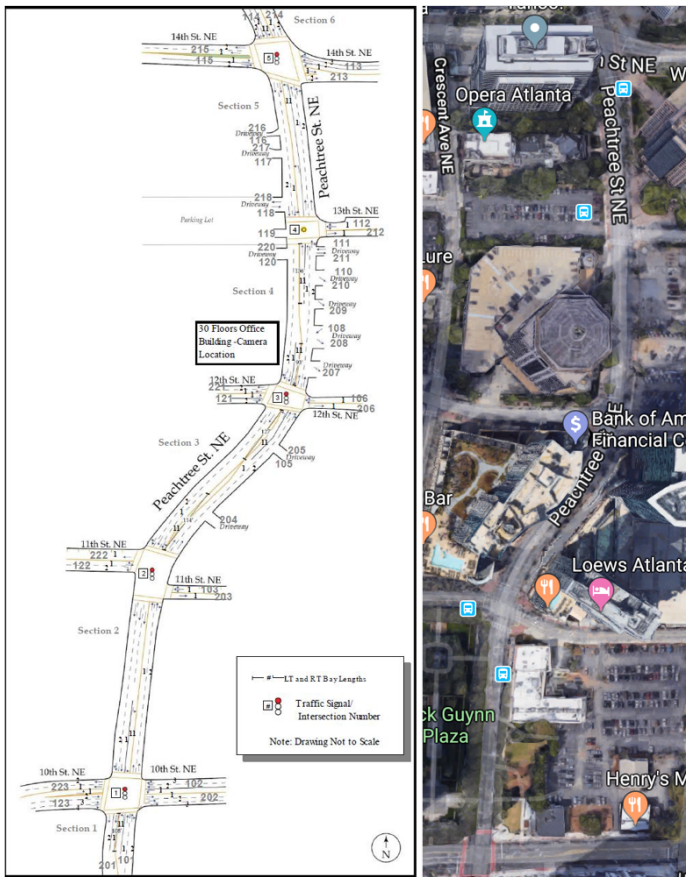

Figure 3: Peachtree street view

The goal of this project is to assess the average travel time for vehicles to traverse a portion of Peachtree Street. Since it is not easy to predict the average travel time physically, it may be important for us to use a simulation model in order to obtain information about the average travel time. To achieve the goal, we could choose a couple of methods such as Process-oriented simulation model. In general, the Event-oriented simulation would be recommended if there is a manageable task for the simple simulation. However, if the simulation is much larger and more complex, the Event-oriented simulation would be much more challenging to handle the simulation because it has to look at all of the code when there are some modifications. On the other hand, a Process-oriented simulation model would be able to shoot the problem.

In this project, we had been trying to implement modeling and simulation process into the specific problem, namely traffic flow at Peachtree street in Atlanta, with three different simulation models under a couple of assumptions. Then, we compared the approaches with respect to inter-arrival times to analyze pros and cons of each method. Moreover, the simulation results were compared against the real empirical distribution generated by NGSIM data.

3 Conceptual Model

3.1 Assumptions

The model mainly focuses on vehicles which were driving NorthBound, going through Peachtree Street between 10th and 14th street. Besides as for Cellular Automata (CA) model, we assume there are two lanes so cars can change lane.

For Event-Oriented and Process-Oriented model, we assume that vehicles drive through each intersection

and section with different constant time. Yet for CA model, the car travel at discrete time stamp.

Specifically, we get the length of each intersection and section from Table 1 and 5 in report Trajectpory Data Description, then we get average velocity of vehicles of each intersection and section based on NGSIMData, then we get constant travel time as follows.

Intersection	1	2	3	4	5
Length(feet)	99.732	129.875	73.513	66.602	121.319
Average travel time(s)	3.2	6.1	3.7	2.0	6.1

Table 1: Intersection information

Section	2	3	4	5
Length(feet)	417.976	412.172	351.511	344.427
Average travel time(s)	23.7	30.5	10.9	41.4

Table 2: Section information

For the traffic light, we use the information provided at NGSIM, which is in the table 3. Pay attention that there is no traffic light at the intersection 3. At the beginning of the simulation, we set all the traffic lights are green, and the it will change based on the light time.

We start simulation at the intersection1, so at the beginning, each car need to consider the traffic light at intersection1. We end the simulation when all the input cars pass through from Peachtree Stree from 10th to 14th.

Intersection	1	2	3	5
Green light time(s)	34.7	51.4	60.9	34.6
Red light time(s)	49.3	55.4	35.7	46.1

Table 3: Traffic light information

3.2 Input and Output

We get our input data in two ways. First, we can use the real data from NGSIM dataset, which means we generate cars based on the inter-arrival time on the intersection1. Second, we can generate the cars randomly at each section and intersection. All the three models use the same input based on these two methods.

For output, we calculate total travel time from at Intersection to Intersections 5. We also count the inter-arrival time.

3.3 Model Limitations

This model couldn't simulate traffic accident scenario. This model mainly focuses on vehicles which were going through Peachtree Street.

For Event-Oriented and Process-Oriented model, vehicles drive through each intersection and section with different constant time, which doesn't match the real situation.

3.4 Mathematical description

3.4.1 Cellular Automata

A regular grid of cells makes up a cellular automaton. Each cell has its state. After an initial state, the new generation will be created by some fixed rules that determines the new state of each cell based the current state of the cell and the states of its neighborhood cells.[1]

1) Single lane model

Let us start from the simplest condition where the road has only one lane.

Road is viewed as an one dimension grid of L cells. Each cell is empty or occupied by a car. Since there is only one lane, so the driver can only looks ahead.

The cars use the following three rules to update their speed[2]:

- IF $v(i) \neq v_{max}(i)$ THEN $v(i) := v(i) + 1$, which means a linear acceleration until the car go to the max speed v_{max}.
- IF $v(i) > gap(i)$ THEN $v(i) := gap(i)$, which guarantees two follow cars will not crash. Here, gap is the width of the gap to the predecessor: $gap(i) = x(pred(i)) - x(i) - 1$
- IF $v(i) > 0$ AND $rand < p_d(i)$ THEN $v(i) := v(i) - 1$, which served as the function for randomly reducing the speed in order to be closer to the real world condition.

2) Two lanes model with changing lanes

In our project, we have two lanes for each direction. Thus, we need to make some modifications based on the above single lane model. The key difference is that cars can exchange line between these two lanes.

Then, let us consider how the driver will change the lane. Based on the common knowledge, we could make the following assumptions for a driver[2].

- He looks ahead if somebody is in your way.
- He looks on the other lane if it is any better there.
- He looks back on the other lane if he would get in somebody else's way.

Based on that rules, we can formulate that a car would change to the other lane if all of the following conditions are fulfilled[2]:

- $gap(i) < l$
- $gap_o(i) > l_o$
- $gap_{o,back}(i) > l_{o,back}$
- $rand() < P_{change}$

Here, similar to the simple one lane model, $gap(i)$ for the number of empty sites ahead in the same lane. Besides, we also define that $gap_o(i)$ for the forward gap on the other lane and $gap_{o,back}(i)$ for the backward gap on the other lane. Meanwhile, $l, l_o, and l_{o,back}$ are the parameters which decide how far the driver look ahead on his lane, ahead on the other lane, or back on the other lane, respectively.

3) Two lanes model with traffic light[3]

In order to consider the traffic light information, we maintain one more list to update the distance from the car to the traffic light, which is s_n in Fig.4.

We use the following rules.

- IF The traffic light is red in front of the vehicle i, THEN $v(i) := \min(v(i), gap(i) - 1, s(i) - 1)$.
- The traffic light is green in front of the nth vehicle:

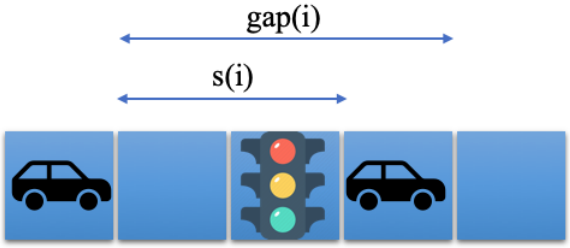

Figure 4: Traffic light description at the intersection for CA model

- IF the next two cells directly behind the intersection are occupied THEN $v_i := \min(v(i), gap(i) - 1, s(i) - 1)$, which could avoid the transition to a completely blocked state.
- ELSE $v_n := \min(v(i), gap(i) - 1)$, which means the car just go as the single lane model.

3.4.2 Event-oriented Simulation

As an option of random generator, we can use exponential distribution to randomly generate the inter-arrival time T of vehicles.

$$P(T \leq t) = 1 - e^{-\lambda t}, t \geq 0 \tag{1}$$

where, λ is the average arrival rate.

Specifically, we can use Inverse Sampling to generate the random interarrival time of vehicles. We first take integration for both sides of equation (1), then we get:

$$F(t) = t + \frac{e^{-\lambda t} - 1}{\lambda}, t \geq 0 \tag{2}$$

Further, we get inverse function as $t = f^{'}(F(t))$. Since the range of F(t) is $0 \leq F(t) \leq 1$, we could randomly generate F(t) with Uniform distribution, then calculate the interarrival time with equation $t = f^{'}(F(t))$.

4 Input Analysis

In this project, we had used NGSIM data, which is a high-resolution trajectory data for vehicles traversing 10th to 14th at Peachtree street in Atlanta, to perform input analysis and to compare with simulation results. For input analysis, we collected the NGSIM data under a couple of assumptions and developed the Python code for post-processing on the data. Based on a few assumptions such as only direction 2, we filtered the data using the Python code as shown in Figure 5.

In [6]:
```
1 NGSIM_traffic_data_filter
```

Assumptions

Out[6]:

Epoch_ms	Local_X	Local_Y	Global_X	Global_Y	Veh_Len	Veh_Wid	...	Org_Zone	Dest_Zone	Intersection	Section	Direction	Movement
1163196100	26.760	1969.251	2230831.683	1377432.360	14.0	7.0	...	102	214	5	0	2	1
1163196200	26.665	1970.030	2230831.488	1377433.131	14.0	7.0	...	102	214	5	0	2	1
1163196300	26.534	1971.031	2230831.200	1377434.165	14.0	7.0	...	102	214	5	0	2	1
1163196400	26.387	1972.233	2230830.860	1377435.507	14.0	7.0	...	102	214	5	0	2	1
1163196500	26.239	1973.573	2230830.539	1377436.774	14.0	7.0	...	102	214	5	0	2	1
1163196600	26.096	1974.890	2230830.217	1377438.041	14.0	7.0	...	102	214	5	0	2	1
1163196700	25.954	1976.184	2230829.896	1377439.307	14.0	7.0	...	102	214	5	0	2	1
1163196800	25.812	1977.482	2230829.575	1377440.573	14.0	7.0	...	102	214	5	0	2	1
1163196900	25.668	1978.802	2230829.254	1377441.839	14.0	7.0	...	102	214	5	0	2	1
1163197000	25.525	1980.080	2230828.932	1377443.105	14.0	7.0	...	102	214	5	0	2	1
1163197100	25.386	1981.268	2230828.611	1377444.370	14.0	7.0	...	102	214	5	0	2	1
1163197200	25.255	1982.422	2230828.290	1377445.636	14.0	7.0	...	102	214	5	0	2	1
1163197300	25.130	1983.610	2230827.969	1377446.901	14.0	7.0	...	102	214	5	0	2	1

Figure 5: Filtered data with the assumptions

Since inter-arrival times at each intersection are not explicitly available from the NGSIM data, we used Google Map(http://maps.google.com) to determine the location of intersections shown in Figure 6, namely a monitoring point, and estimate inter-arrival times at each intersection.

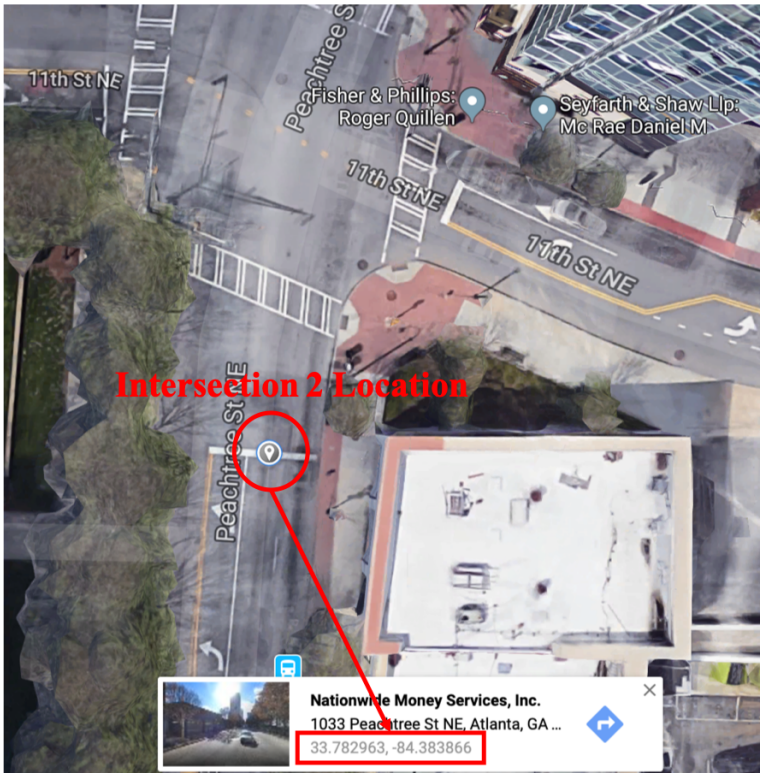

Figure 6: Intersection 2 location estimation using Google Map

Since the Google map provides latitude anw longitude coordinate; however, the NGSIM data defines

global locations with x and y coordinate, we used the following website(http://www.earthpoint.us/stateplane.aspx) to convert from latitude/longitude to x/y coordinate.

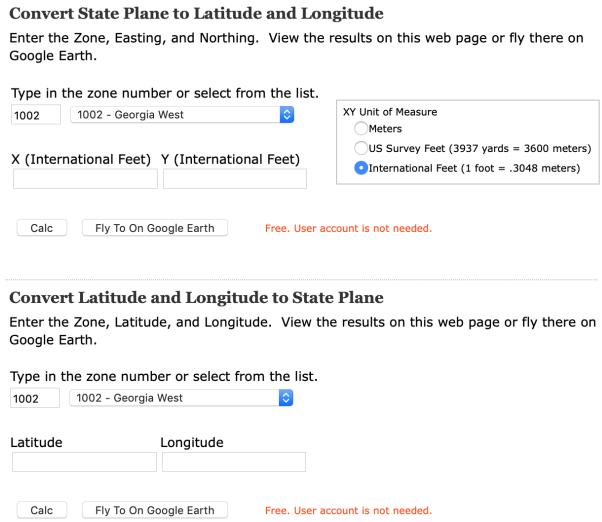

Figure 7: Conversion from lat/long to x/y coordinate

Using the location information, we implemented a method in the Python code where it tries to find the closest point of vehicle to the monitoring point. Finally, inter-arrival times at each intersection were calculated based on arriving times of each vehicle at the particular monitoring point. Each intersection was investigated for the inter-arrival times and they are shown in Figure 8.

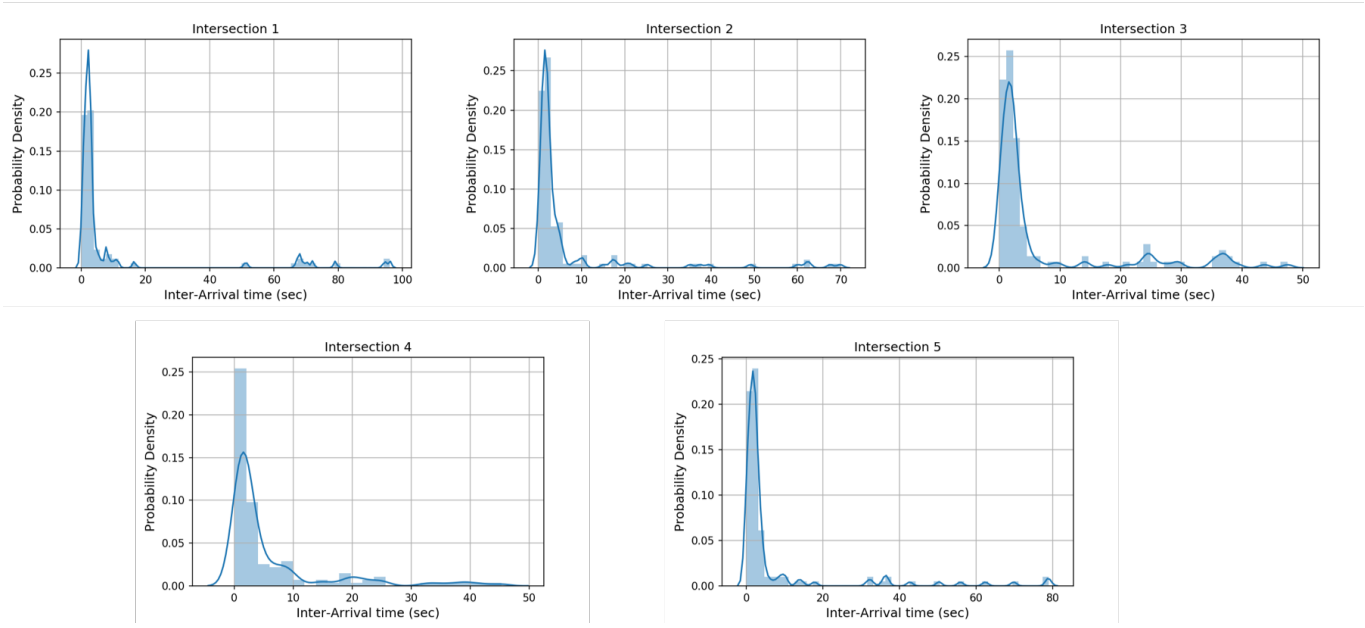

Figure 8: Inter-arrival time results at each intersection

5 Simulation Model

5.1 Cellular Automata

1) Initialization for cars and road

For the road length as well as the number of cells, we also use the data from the real dataset. We show the sections and intersections length in Table 1 and 5. Thus, the total length for the simulation is 2017.127 feet. The average car length from the NGSIMData is about 16 feet, thus we have 2017.127/16 = 126 cells with each cell is 16 feet.

For the cars, we use the same method for all the three models, which is shown at Section 3.1. Besides it, we random give the lane position that is west or east side. Then, for each car, we give them a random speed that smaller than the max speed v_{max} of the road. From NGSIMData, $V_{MAX} = 55.82$ ft/s. Thus, for CA model, where v_{max} represents how many cells a car could move forward at each time stamp, we set v_{max} as a round number for 55.82/16 so it is 3. One time step approximately corresponds to 1 second in real time due to the cell length.

2) Simulation Steps: Iteration update

After initialization, we have the initial state for each cell. Then, we give the number of iteration times and update the state based on the above discussion rules.

For each iteration, we have the following steps:

- Step1: Calculate parameters.

  For this part, we will first let the cars speed up linearly for the maximum speed with the rule 1 for single lane model. Then, we will get gaps between current car and the front one with the equation $gap(i) := x(pred(i)) - x(i) - 1$. Finally, we will compute the forward gap as well as the backward gap on the other lane, which is $gap_o(i)$ and $gap_{o,back}(i)$ defined before.

- Step2: Determine state based on traffic lights. We round the time of traffic lights to integer and apply the rules in Section 3.3.1. Meanwhile, we set the four traffic light at four cells based on their global location. Since we start at intersection1, so the four traffic lights are at cell 1, 35, 58, 107.

  For example, if a car is just one cell behind the traffic light, and the light is red, so $s(i) = 1$, then it speed will be $\min(v(i), gap(i) - 1, s(i) - 1)$ which is 0 in this case.

- Step3: Switch lanes.

  Now, we use the above four rules to decide whether a car will change the lane or not. For simplification, we use the fixed number for l, l_o, and $l_{o,back}$ to 1, which means the driver will definitely change the lane if the road condition is satisfied.

- Step4: Random slow down.

  We implement this part by the rule 3 for single lane model.

- Step5: Move forward.

  For a car i, if the updated $v(i) > gap(i)$, then the new position of this car is the current location in addition to $gap(i) - 1$. Else, the new location is the current location in addition to $v(i) - 1$.

  With all the above state change steps, we can now get the new location of each car as well as the speed of each car.

5.2 Event-Oriented Simulation

Event-oriented simulation focuses on events and event computations. The simulation is viewed as a sequence of event computations, and each event has a time stamp indicating when it occurs.

In addition, simulation program of Event-oriented is written as a set of event handlers, and the simulation uses Future Event List to advance simulation time. Every event computation would schedule new event added to the Future Event List(FEL). Further, through event processing loop, the smallest timestamped event could be removed from FEL.

To setup Event-oriented queueing model, we first define the Event types, State variables and Constants as follows.

Event types
Arrival: vehicle arrives at intersections 1, 2, 3, 5
Red traffic light: traffic light changes from Green to Red
Departure: vehicle leaves intersections 1, 2, 3, 5
State variables
StopAtRedLight: vehicle stops at entrance of intersection when traffic light is Red
TrafficLightStatus: traffic light status of intersection 1, 2, 3, 5
VehicleLocation: vehicle current location
Constants
InterS(k): travel time of Intersection k
S(k): travel time of Section k

Figure 9: Event types, State variables and Constants

Then we can setup the world view of Event-oriented model as follows.

Event handler procedures
ArrivalEvent {processing current arrival event with vehicle location, vehicle index and traffic light status, then schedule future event and add it into FEL}
RedTrafficLight {check traffic light status based on current event, update traffic light indicator and corresponding time value, update car queue, then schedule future event and add it into FEL}
DepartureEvent {processing current departure event with vehicle location, vehicle index, then schedule future event and add it into FEL }
Other defined Function
AdvanceTime {calculate future event time based on current time and corresponding constant travel time InterS(k) or S(k)}
VehicleLocation {calculate vehicle index and vehicle location based on current event}
PriorityQueue {keep the FEL in ascending order }
Event processing loop
While(FEL is not empty) { E = smallest time stamp event in FEL Remove E from FEL Now := time stamp of E call VehicleLocation call event handler procedure}

Figure 10: World view of Event-oriented model

As above figure shows, after the input(arrival time of vehicles) was added into FEL, simulation program enter into event processing loop. The smallest time stamp is removed from FEL and set as current event.

Through VehicleLocation function, the vehicle index and vehicle location could be found based on current event, then the corresponding event handler would be called. The specific process is shown as follows.

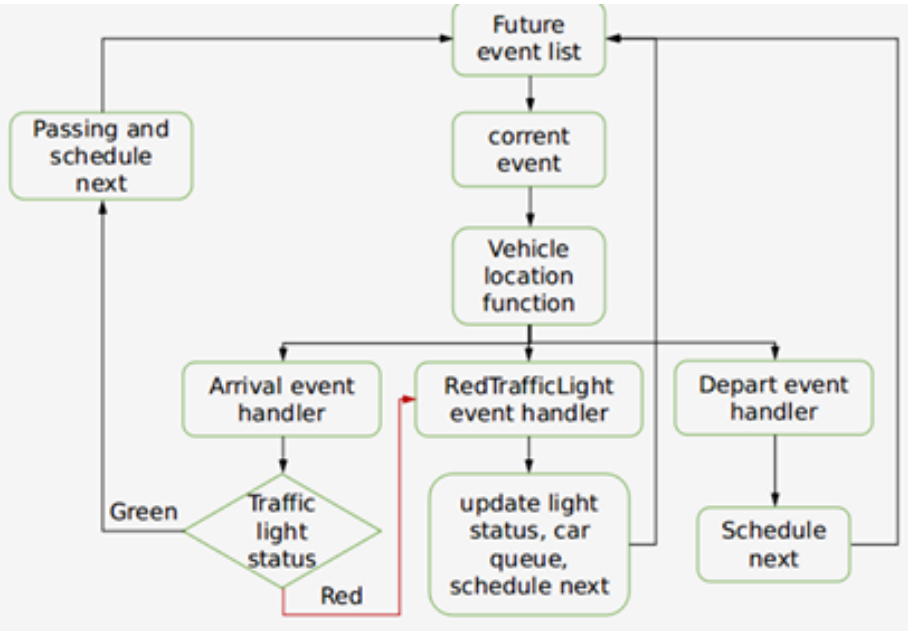

Figure 11: Diagram of processing event of simulation program

As above figure shows, in the RedTrafficLight event handler, car queue is used to record all the vehicles which are waiting for the next Green light. When the traffic light changes to Green, vehicles will leave the queue one after another, which makes their next arrival time different.

5.3 Process-Oriented Simulation

In general, a process-oriented simulation is intended to model a specific entity in the simulation and the entity is described by a process. For this project, we considered vehicles as a process and modeled them as a thread in the simulation. In terms of the vehicles, we implemented a function, namely WaitUntil, to suspend the process while simulation time still advances. To be more specific, due to the function, we were able to take account for the communication between vehicles and traffic light systems at each intersection. The schematic of this implementation is shown in Figure 12.

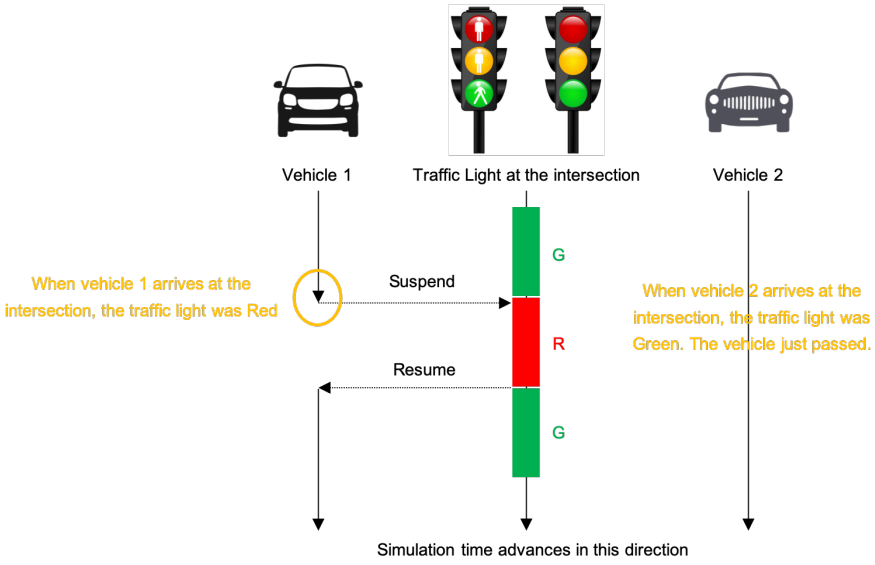

Figure 12: Thread communications in the model

As can be seen in Figure 12, the traffic light is considered as a resource in which vehicles shared the resource to complete their processes. For this project, we defined four resources to take into account four different intersections. This is because each intersection has different signal timing information. In order to propagate vehicles in the simulation, we used both the empirical distribution with respect to inter-arrival time and a distribution generated by random number generator as an input at intersection 1. For example, the empirical distribution at intersection 1 is shown in Figure 13.

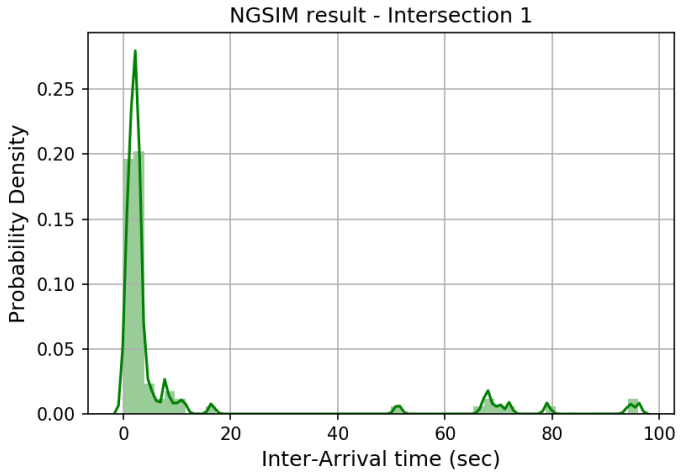

Figure 13: Inter-arrival empirical distribution at intersection 1

To generate the inter-arrival distribution, we assumed the following statements: 1) Vehicles are only able to go straight. No turn left or right is allowed. 2) All vehicles depart from right before intersection 1. It means the traffic light at intersection 1 was considered in the simulation. 3) There are either green or red light for traffic light at all intersections. We did not consider yellow light in the simulation. 4) When the traffic simulation starts, all traffic light at intersections are initialized with green light. 5) Based on the data, we did not specify traffic light at intersection 4. 6) North bound is only considered in the simulation. 7) The lengths of section and intersection are defined as constant values.

Based on the assumptions, the process-oriented simulation was supposed to be performed with 90 vehicles (or processes) and 4 traffic light resources. The first vehicle departed from the starting point (before intersection 1) and suspended its process whenever it realized the resource is not free to go through. In the meantime, the simulation process could resume the execution at exactly the point at which it had been stopped. As such, we were able to calculate inter-arrival times at all intersections. For example, the comparison between simulation results and real data distribution at intersection 2 is shown in Figure 14.

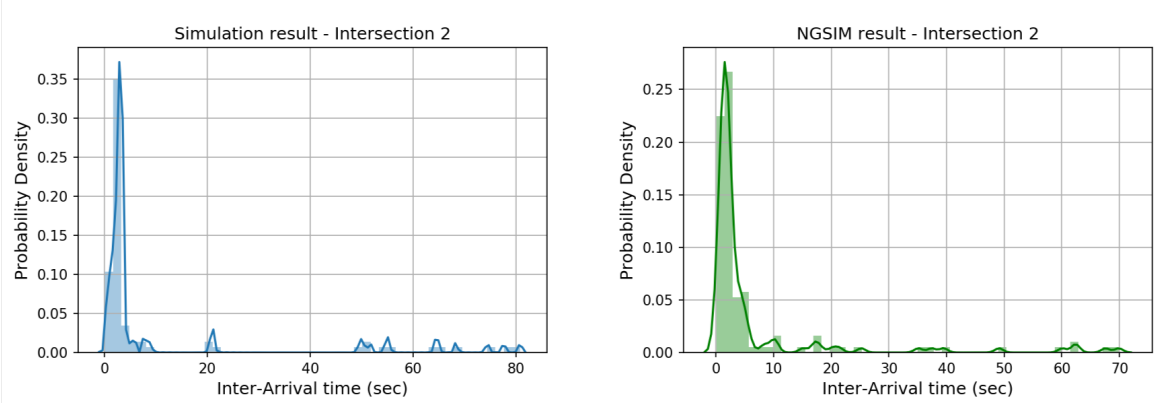

Figure 14: Simulation vs. Real data w.r.t. inter-arrival time at intersection 2

Lastly, we implemented a logic (we will shortly discuss about it) and a queue incorporated in 'simulation process' function in the Python script for process-oriented simulation model. This was needed because it was observed that some vehicles had exactly same arrival time at the intersection. This was because they stopped at the same point when they met traffic light with red. As soon as the traffic light became from red to green, they were supposed to arrive at the next point with exactly same time information. To resolve this issue, we created a queue at each intersection and physically formed the vehicles in a line. As such, we were able to calculate the time difference between two vehicles and added some additional times depending on how many vehicles are stored in the queue. This implementation is described in Figure 15.

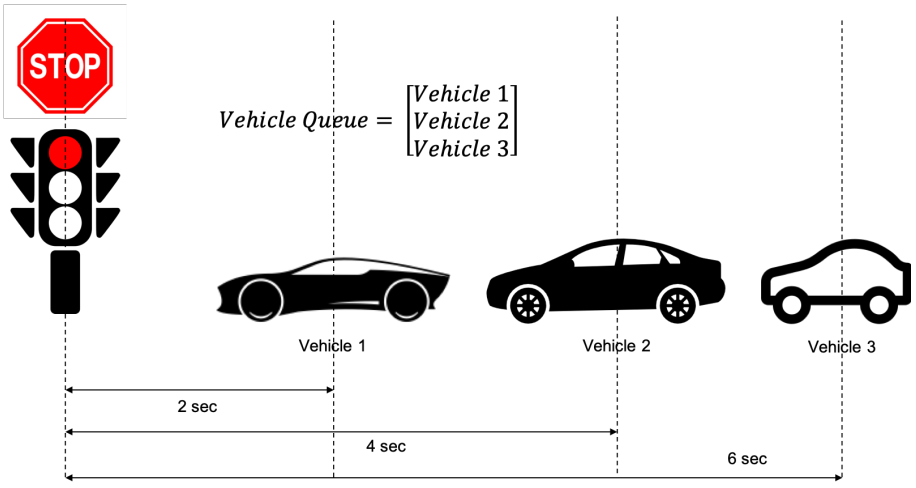

Figure 15: Trouble shooting to reduce the gap between reality and simulation w.r.t. STOP waiting time

6 Verification and Validation

6.1 Verification

In general, verification process is to ensure that the computer program implementation is correct. A question related with the verification process would be 'Did we build the model right?', which is concerned about the correctness of the simulation program developed. For this project, we performed the verification process for each simulation model as follows.

6.1.1 Cellular Automata

I took the first 50 cells and 5 iterations as examples. The yellow blocks are the car and we have 2 lanes. You could find that the cars are moving forward and are changing lanes.

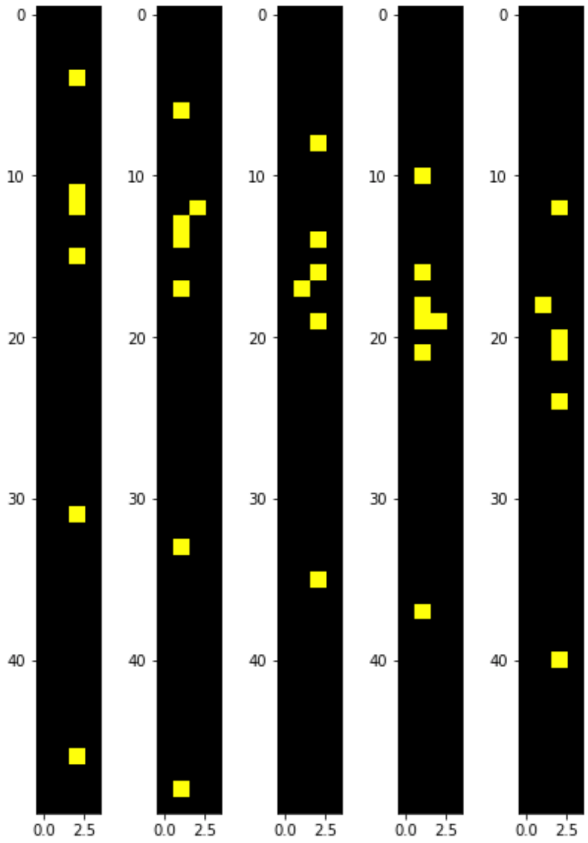

Figure 16: First 50 cells for No.1 to No.5 iterations simulation visualization result

6.1.2 Event-Oriented Simulation

To do verification of the developed simulation software, two different dataset are used to test the correctness. First, dataset with initial arrival time of 1 vehicles. Second, dataset with initial arrival time of randomly generated 200 vehicles.

For the first dataset, the state variables status and output could be used to verify the correctness, which includes processed event, Future event list, vehicle location, car queue, traffic light status.

For the second dateset, it is difficult to verify the above information. So, we can count number of arrival event at intersection 2, 3, 5.

If those information are correct during every iteration, and those number equal to the number of initial arrival time, then correctness of the software could be verified.

The state variables status and output during every iteration under first dataset are shown as follows.

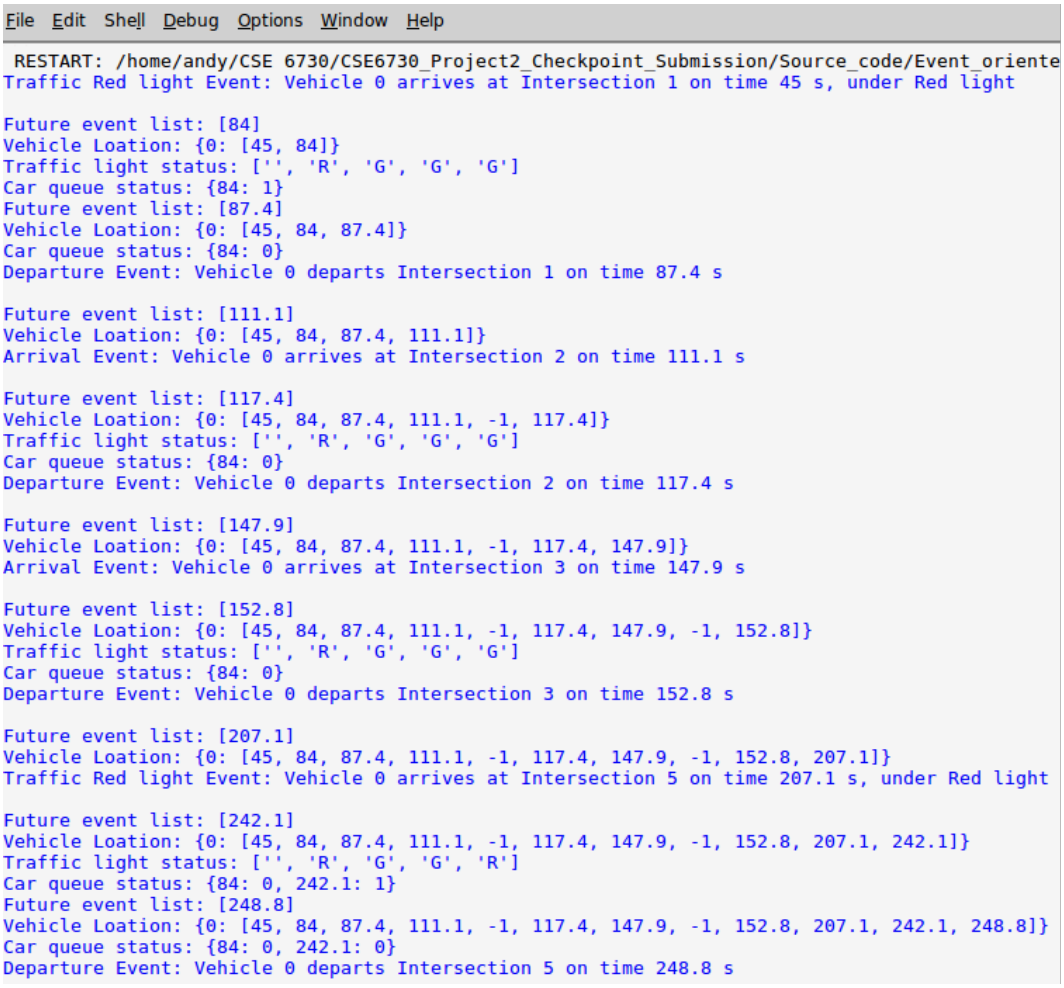

Figure 17: Verification result based on first dataset

As the above figure shows, at last step, Vehicle Loation includes 12 numbers in which each intersection(1,2,3,5) has 3 numbers to represent arrival, red light and departure time. If no red light, then the value is set as -1.

In addition, vehicle arrives at intersection 1 on 45s under red light, then traffic light status at intersection 1 changes to Red. Further, vehicle should wait for next green light, that is 84s, which is added into FEL. Car queue shows that 1 car should wait until time 84s. And after 84s, no vehicle is in the car queue, and future event is added into FEL. The program repeats this process until vehicle departs intersection 5. In summary, We can see that every iteration outputs correct answers.

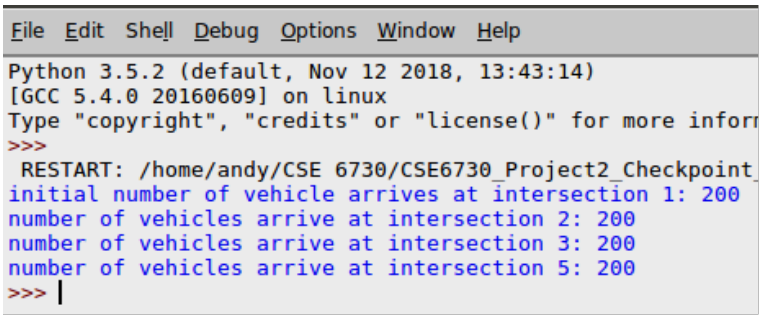

Figure 18: Verification result based on second dataset

As the above figure shows, numbers of arrival event at intersection 2, 3, 5 equal to the initial number. Based on the above results, the correctness of the software can be verified.

6.1.3 Process-Oriented Simulation

It is well known that there are many different methods to verify developed simulation models. For a process-oriented simulation model developed for this project, we employed one of static verification techniques, which is to walk through the program structure to see if it is working reasonably. First of all, we implemented a random number generator with heap priority queue structure to make sure that vehicle time events are generated in time stamp order. The verification result in terms of the heap priority queue structure is shown in Figure 19.

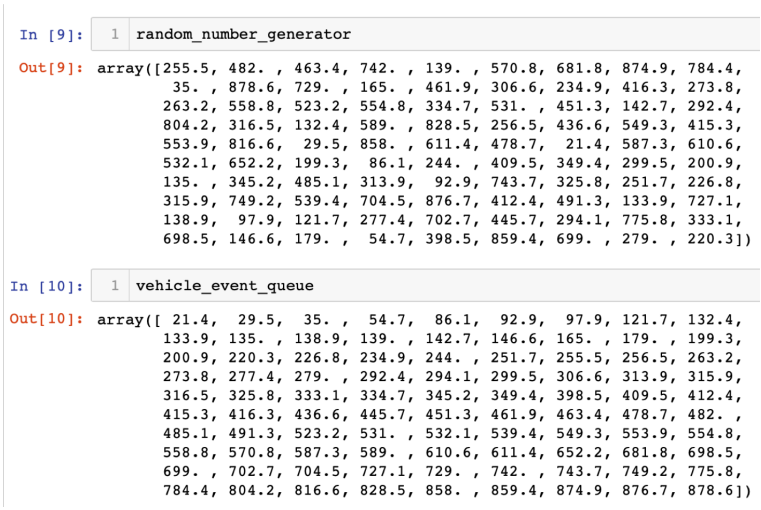

Figure 19: Heap priority queue for random input generator

In terms of traffic signal, since we implemented four different threads for each intersection, we checked if each intersection thread has different signals for green and red. For example, the following Figure 20. describes signal timing and color for intersection 1.

Figure 20: Intersection 1 traffic light signal timing and color in the simulation

In terms of interaction between vehicles and traffic light at all intersections, if the vehicle runs into Green, the algorithm advances the simulation time and updates vehicles time information. On the other hand, if the vehicle runs into Red, the algorithm let the vehicle stop at the intersection and resume the execution until the resource is free. In order to verify the interaction between vehicle and traffic lights, we intentionally changed the traffic lights from Green to Red. Finally, it was observed that the vehicle stopped at the intersection 2 and 3, which is shown in the Figure as following.

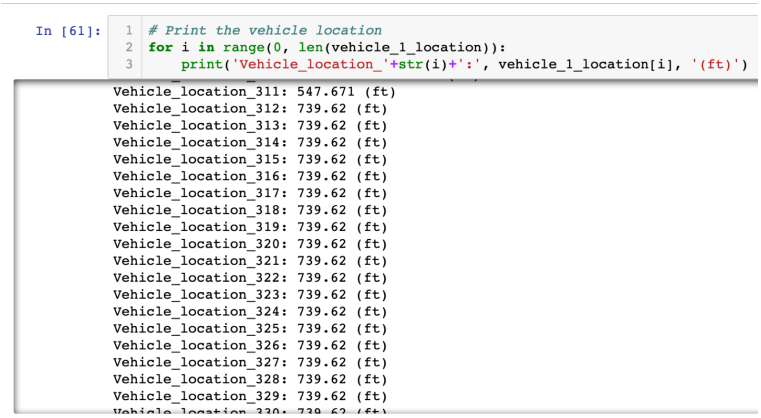

Figure 21: Results for the vehicle simulation with red light

6.2 Validation

We compare our simulation results with the information we extract from the NGSIM dataset.

6.2.1 Inter-arrival time

We show our simulation results about inter-arrival times against the empirical distribution achieved from input analysis. All our simulation results have already delete the warm-up period, which we will explain in details in Section 7.2.

From the following figures, Process-oriented and Event-oriented models have quite similar results, which proves that our models are efficient and robust. CA is a little different, and we will analysis it in the Section 8 later. Overall, all the three models have same trend for the inter-arrival time compared with real data set, which gives a good validation.

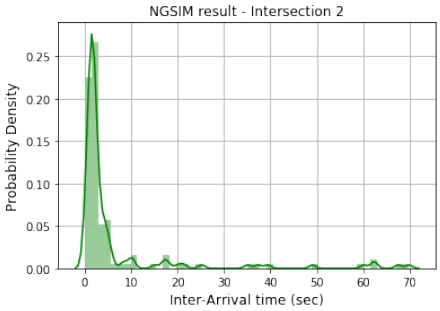

Figure 22: NGSIM inter-arrival time distribution at intersection2

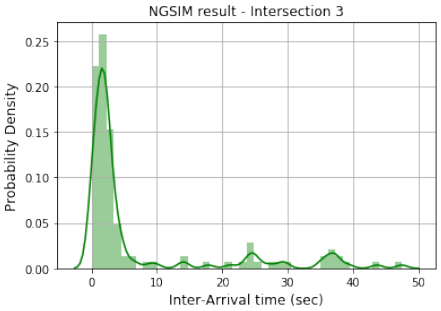

Figure 23: NGSIM inter-arrival time distribution at intersection3

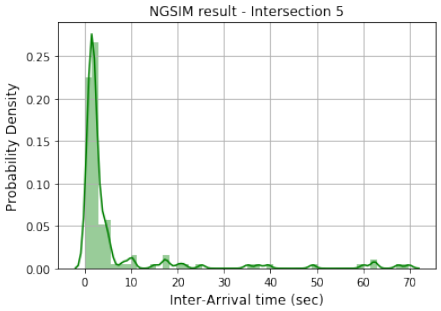

Figure 24: NGSIM inter-arrival time distribution at intersection5

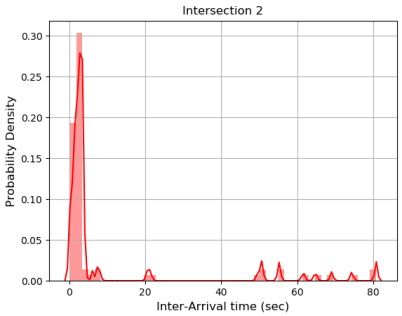

Figure 25: Event-oriented inter-arrival time at intersection2

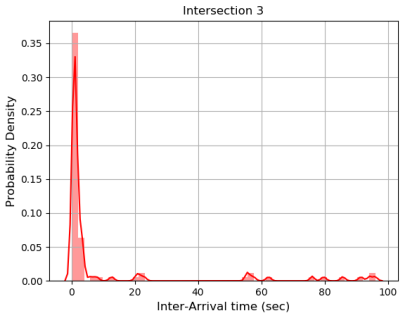

Figure 26: Event-oriented inter-arrival time at intersection3

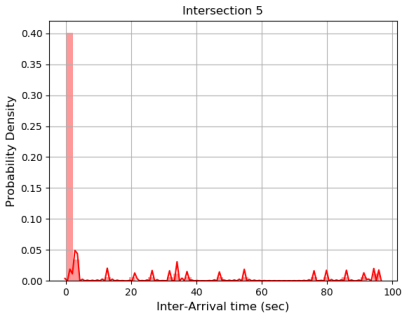

Figure 27: Event-oriented inter-arrival time at intersection5

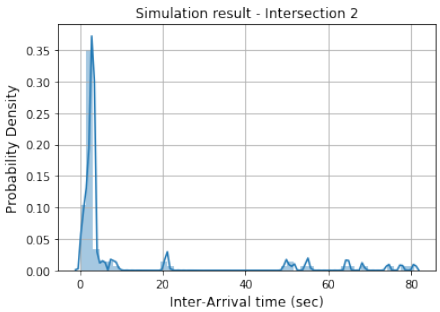

Figure 28: Process-oriented inter-arrival time at intersection2

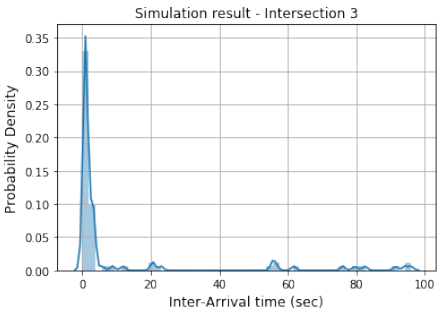

Figure 29: Process-oriented inter-arrival time at intersection3

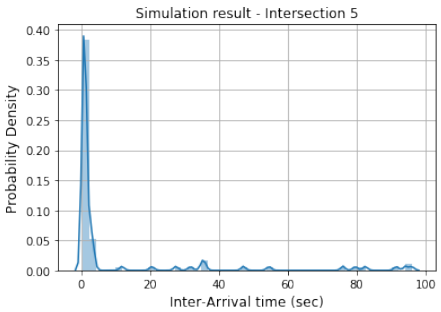

Figure 30: Process-oriented inter-arrival time at intersection5

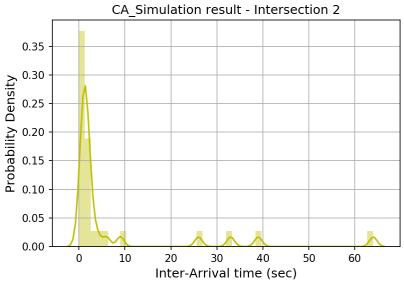

Figure 31: CA model inter-arrival time at intersection2

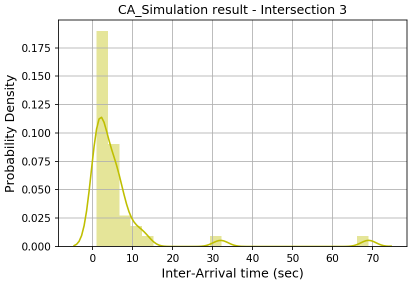

Figure 32: CA model inter-arrival time at intersection3

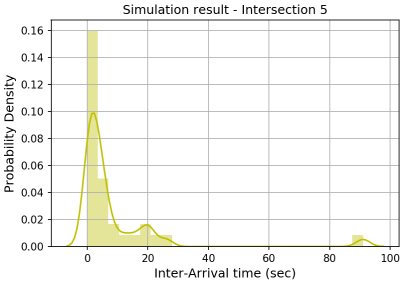

Figure 33: CA model inter-arrival time at intersection5

6.2.2 Total travel time

From the real dataset, the average speed for the car which under our one direction assumptions and pass through from intersection 1 to 5 is 140.41 seconds. The distribution is in Fig 34.

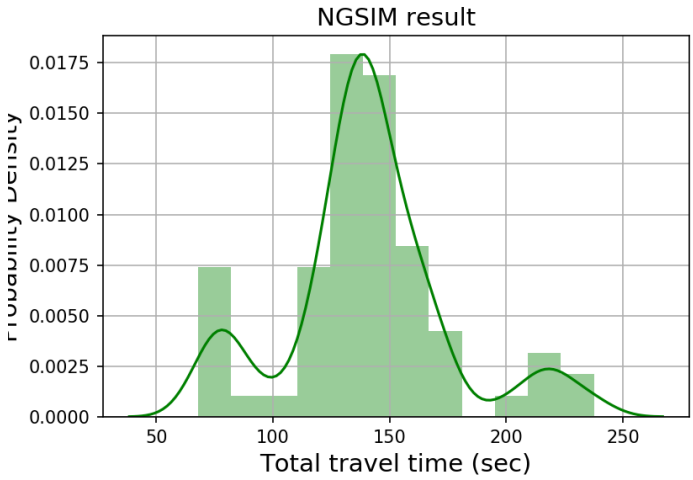

Figure 34: NGSIM dataset total travel time

We run each simulation model for 20 times and get the total travel time for each car, and then take the average to get the average total travel time for each model. The results are in Table 4.

Model name	Event-oriented	Process-oriented	CA
Average total travel time (s)	193.6	195.0	137.5

Table 4: Average total travel time for each model

We show the travel time for each car at one time simulation. We will analysis the possible reasons for the difference at the discussion section.

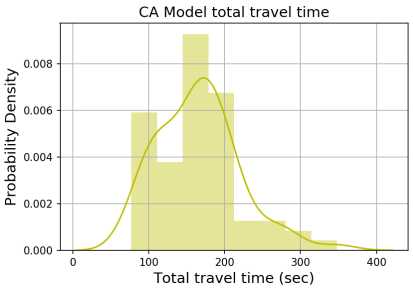

Figure 35: Total travel time for CA model

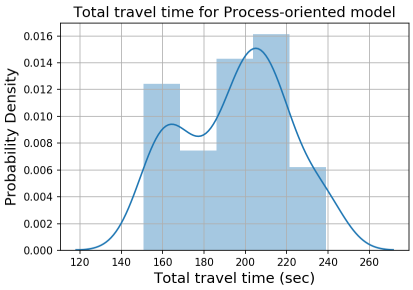

Figure 36: Total travel time for Process-oriented model

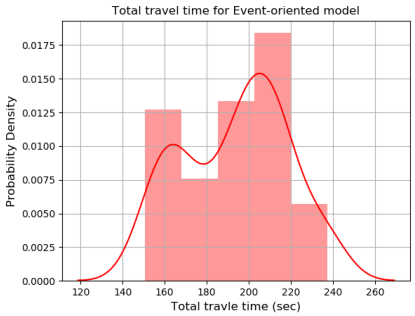

Figure 37: Total travel time for Event-oriented model

7 Output Analysis

7.1 Design of Experiment

First of all, we wanted to see the effect of levels of traffic intensity on the simulation. In order to achieve the goal, we created different settings such as from 20 to 200 vehicles; and see if how average total travel time changes as the number of vehicles increases. The result is shown in Figure 38.

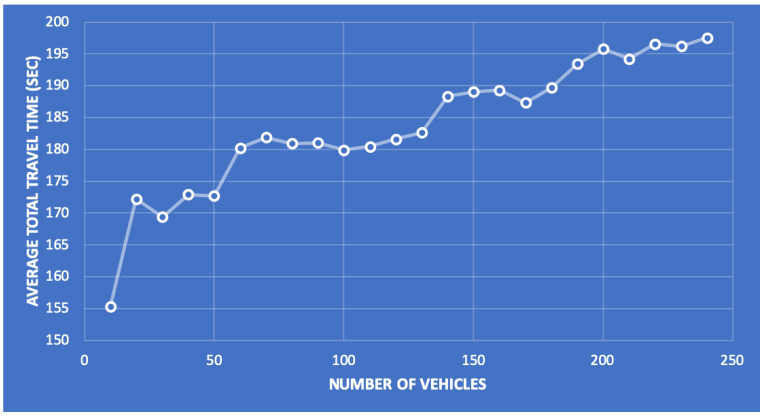

Figure 38: Total average travel time vs. Number of vehicles

As can be seen, it was observed that the average total travel time increases as the number of vehicles increases. Second, since the process is stochastic due to the random number generator, we created the following Design of Experiment table and performed multiple simulation runs in order to calculate average total travel time with different models.

Model	EX1	EX2	...	EX20	avg
CA	131.5	179.2	151.5	136.0	
Event-oriented	180.2	186.9	...	182.4	184.1
Process-oriented	190.1	182.4	...	186.3	186.3

Table 5: Average total travel time with different model

7.2 Random Generator VS Empirical distribution

We show our simulation results about inter-arrival times and total travel time based on random generator, and make comparison with the empirical distribution achieved from input analysis.

We just show one example of our three models. For CA model, we show the results with the same cars of empirical distribution that is 91. The results is a little different as the empirical distribution but the trend is similar. The total travel time is very similar to uniform distribution, which is reasonable.

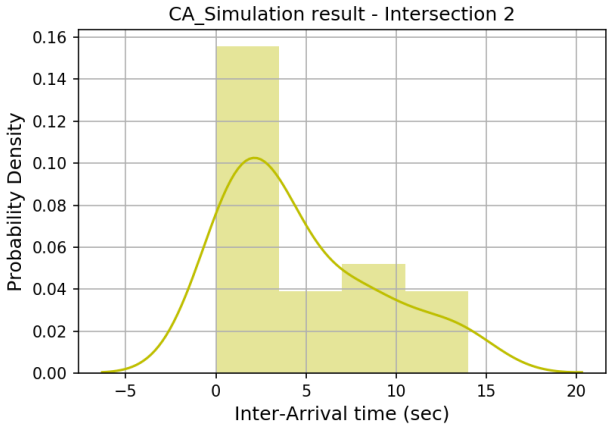

Figure 39: CA model inter-arrival time at intersection 2 with uniform random generator

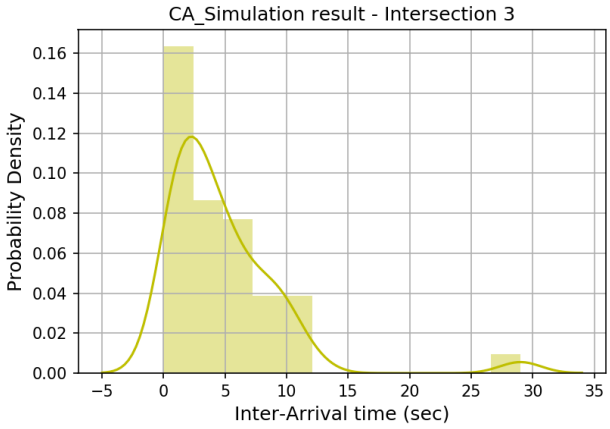

Figure 40: CA model inter-arrival time at intersection 3 with uniform random generator

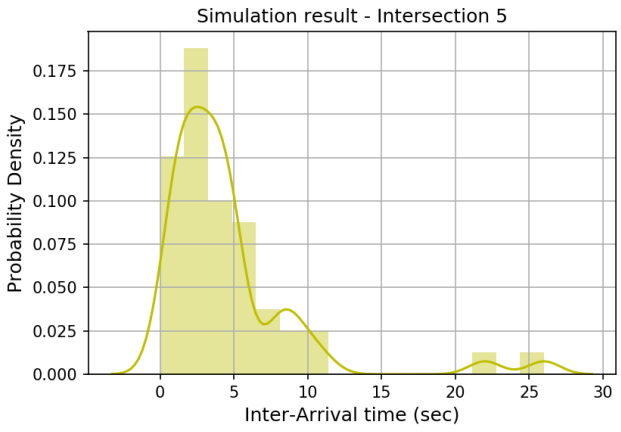

Figure 41: CA model inter-arrival time at intersection 5 with uniform random generator

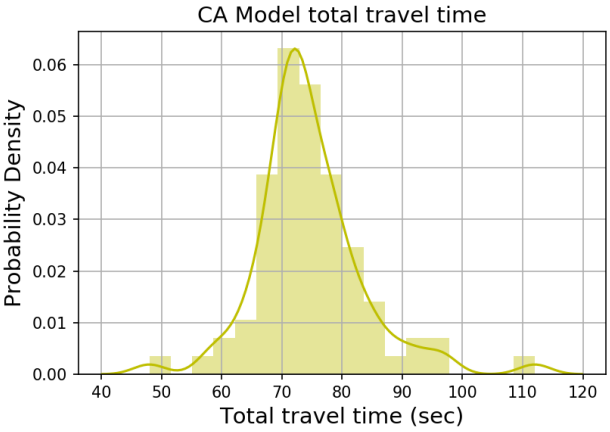

Figure 42: Total travel time for CA model with uniform random generator

7.3 Warm-up Period Analysis

We count how many cars are on the street at each simulation time step to find when the warm-up period will end. Fig. 43 shows the whole process for one time CA model simulation, and it takes about 1000 s. We find it is periodic, which makes sense due to the input inter-arrival time has several very large number 70s and 80s; meanwhile, some cars will exit when some cars still enter.

Then, we take the first 100s shown in Fig. 44. After about 22s, the car number become stable so we choose the 22s as the transition time point for CA model. Due to our input inter-arrival time, it means

after the first 10 cars entering the road, the traffic become stable.

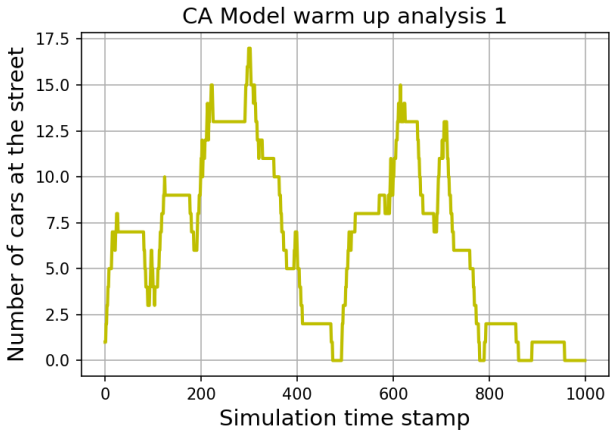

Figure 43: Car number with simulation time stamps for CA model

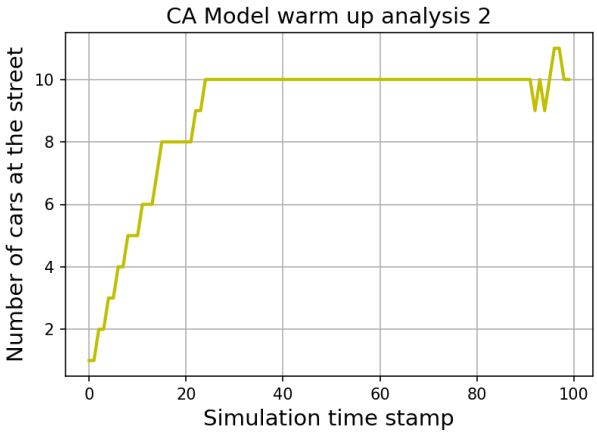

Figure 44: Car number with first 100 simulation time stamps for CA model

8 Discussion

8.1 The results difference from three models

For Event-Oriented and Process-Oriented models, the results at validation part are exactly the same, which proves the accuracy of our models since they are supposed to be similar.

For CA model, the assumption is different since it use discrete time and cells, so the results is a little different. Meanwhile, CA model has random slow down so the results will be different for each experiment. However, the overall curve have the same tendency which illustrates the model is reasonable.

8.2 Total travel time

For Event-oriented model, the total travel time is a little bit different with the real data set. The possible reasons are as follows: this model assume that vehicle takes different constant time traveling through intersection and section. In addition, all the vehicles start at intersection 1, no vehicles start at intersection 2 or 3. If this model is capable of processing vehicles traveling at varied velocity and starts at intersection 2 or 3, the results would much close to the real situation. Meanwhile, we assume all the traffic light begins at the green, so the light timing may be different from the real data set.

For CA model, the total travel time is similar to the real data set, while each simulation result will be different due to the random slow down at CA model. Besides, if we use a simpler rule for red traffic light, which assume all the cars besides the traffic light will stop at the same time step, the average of total travel time becomes larger and the value is about 159.4 seconds. It shows the rule we are using is more reasonable.

9 Conclusion

In this project, we successfully implement three models to simulate the traffic. We consider each steps of Modeling and Simulation (M&S) life cycle to make a complete and reasonable project.

we give detail explanation for conceptual model including assumptions, input and output of the simulation which is consistent for all our three models. As for each model, we give specific statements for mathematical foundations and simulation steps.

Then, based on our deep analysis for input data set, we do the verification and validation part to prove the efficiency and accuracy of our models. Also, we do the output analysis including warm-up period and random generator. The validation shows results for inter-arrival time

In the future, we would like to consider more complex real world conditions. Meanwhile, we could design a user-friendly method to show the output.

References

[1] Tommaso Toffoli and Norman Margolus. *Cellular automata machines: a new environment for modeling.* MIT press, 1987.

[2] Marcus Rickert, Kai Nagel, Michael Schreckenberg, and Andreas Latour. Two lane traffic simulations using cellular automata. *Physica A: Statistical Mechanics and its Applications*, 231(4):534–550, 1996.

[3] Elmar Brockfeld, Robert Barlovic, Andreas Schadschneider, and Michael Schreckenberg. Optimizing traffic lights in a cellular automaton model for city traffic. *Physical Review E*, 64(5):056132, 2001.