N

| For office use only | Team Control Number | For office use only |
|---|---|---|
| T1 _____ | 66668 | F1 _____ |
| T2 _____ | | F2 _____ |
| T3 _____ | Problem Chosen | F3 _____ |
| T4 _____ | B | F4 _____ |

# 2017

**MCM/ICM**

**Summary Sheet**

(Your team's summary should be included as the first page of your electronic submission.)

Type a summary of your results on this page. Do not include the name of your school, advisor, or team members on this page.


We determine the optimal design of toll plaza by maximizing throughput and accident prevention after minimizing the initial construction cost. There are three models, two fundamental and one comprehensive, to examine the problem from different perspectives.

We treat the toll plaza as a 2-D figure and cars as identical rectangles moving on it. In our first model, we determine an optimal shape and size which minimizes the cost. The second model explores the relationship between the throughput and several variables such as the proportion of autonomous cars. We then developed model 3 based on cellular automata. Using information from model 1 and model 2, we further take accident prevention into consideration. With the help of MATLAB, we get an optimal solution.

An optimal solution can always be reached. The shape and size changes as the conditions of the cars changes. The merging pattern changes as the proportion of autonomous cars changes. Both model 2 and model 3 agree in the results of the experiments they have in common.

Although we introduce many variables, we don't involve any real-world data. We also attach our codes for the programs used in the appendix.

Dear sir/madam,

We are writing to happily announce that our team has determined the best design of toll plazas on the lanes. We sincerely hope we can cooperate as soon as possible to apply the design into real life in order to save your company's money as well as promote traffic efficiency. In the following paragraphs, we are going to show you some general ideas of our design and hope it will interest you.

To determine the shape and size of the toll plaza is never an easy question. We take various factors into consideration, for example, the cost of building the plaza, the throughput, and the preventions of accidents. Besides, the arrangement of the different kinds of tollbooths according to different situations are mentioned in our design. On account of the future development of autonomous cars, our team also examined the effect of the increasing number of self-driving cars on the road.

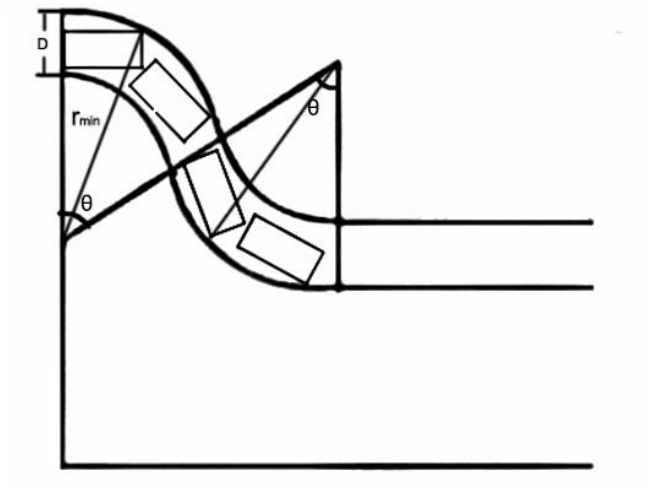



Figure 1

First of all, we decided the best shape of the plaza based on mathematical calculation and the usual minimum turning radius of cars. By computer analysis, we found that the shape of the design should be as in Figure 1 if our goal is to minimize the area and save money.

After that, we simulated the traffic situation using computer programs and got some really inspiring results. First of all, after we take throughput into consideration, the shape of the toll plaza should still be the same as figure 1, because it maximizes the amount of throughput. Besides, it is not efficient to build a lot more tollbooths than the lanes, especially when there is a long-term heavy traffic. Our calculation shows that if the shape of the fan-in area doesn't change, building more tollbooths will increase the throughput if the traffic is light, and it cannot reach this goal if there is a consistent heavy traffic due to several reasons.

Also, according to our conclusions, we don't encourage all drivers to always

follow the traffic rules, especially when there is little traffic. However, if the development of autonomous cars is promising and all cars will strictly follow the rules set to them in the future, then it is still recommended that cars still stick to the traffic rules. These two conclusions may seem controversial, but actually they both can be explained logically.

All in all, we have reached very intriguing conclusions and we can promise that this is going to change the world of toll plaza building.

Looking forward to your reply and our future cooperation.

Yours,
Team 66668

# A Best Design of the Toll Plazas

## Contents

# 1 Introduction

If there is anything that takes both your time and money during a trip, it must be driving through a toll station. Low passing efficiency and high initial construction cost of the station have made the construction of toll plazas a stubborn problem. Needless to say, safety is another significant factor that needs to be taken into consideration. We analyze different layouts of the toll plazas and decide on an optimal solution.

There is more than one way to explain what an optimal solution is. Different people may hold different opinions about how many weights should be given to throughput, accident prevention and cost. Instead of concerning ourselves with such questions, we choose to define an optimal solution as following:

- An optimal solution is a solution with minimized cost where throughput and accident prevention are as high as possible.

That is to say, we put cost in the first place.

There are many previous works that are related to this problem. However, most of them focus on the design of tollbooths instead of the fan-in region. Still, many of them are useful for us. For example, [1] provides a good design of tollbooths such that the accident prevention is achieved to a large extent. We will use the results to design an optimal fan-in region. [2] describes a fact that drivers tend to choose a toll lane at the same side of the toll plaza from which they exited the span. Based on this, we assume that the plaza is symmetric so that we only consider half of it.

We will construct three models to deal with the optimizing problem. The first will mainly analyze how the layout of the plaza affects the construction cost. The second will deal with the throughput part. And the third will be comprehensive and also include the accident prevention part.

# 2 Assumptions and Justifications

1. **The station is symmetrical** According to some references such as [2], most drivers would like to choose a tollbooth on the same side (left or right) of the toll plaza with themselves. Hence it is reasonable to assume the plaza is symmetrical and consider only half of it. Moreover, if the station accepts cars from both directions, we can still apply our model with few changes.

2. **The probability of a car choosing to pass each tollbooth is the same.** This assumption is valid because the fan-out area is left out of our consideration. In real life, it is more realistic when the traffic is heavy.

3. **All cars are identical in shape and size.** This is a simplifying assumption.

4. **The following assumptions are about autonomous cars:**

   - **Autonomous cars will always follow the traffic rules.** In order to distinguish autonomous cars from human-driven cars, we set the error rate of self-driving cars to zero.In contrast, the error rate of human-driven cars is always non-zero in our models.
   - **All self-driving cars will choose electronic toll collection booths** This is another criteria which tells the difference between self-driving cars and human-driven cars. In particular, the rate of human-driven cars with transponders for ETC booths is set as a parameter in our model.

# 3 The Model

There are three different models to deal with the problem. The first model is about minimizing the cost. The second model is about maximizing the throughput. The third model is a combination of the first two and is more complicated.

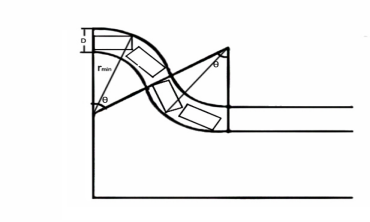
### 3.0.1 Model 1

In this model, we consider the physical conditions of cars and design a possible layout of the fan-in region. We calculate the area of the region in order to minimize the cost of the construction.

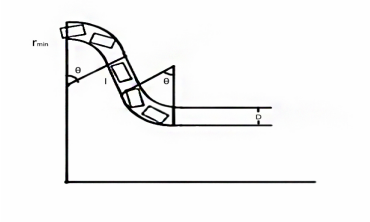
There is a general assumption in this model: A car will always need a lane with a width of $D$ to move, which is valid since cars have to keep a distance from each other.

After some rough search on the Internet, we notice that all cars have their own turning radius, which is the radius of a smallest circle it can make. Therefore, we develop our first model by drawing the moving path of a car

leaving from the outermost tollbooth. Since it should be most difficult for this car to make its turn, as long as it can successfully move into the outgoing lanes, cars on other roads should all be capable of merging into the lanes. Hence, we suppose that the car will try to turn the moment it leaves the tollbooth, because this will reduce the total area of the fan-in region. After turning for a certain angle $\theta$ , it changes its direction immediately to turn into the outgoing lanes(see Figure 3.1.1).



(a) Figure 3.1.1



(b) Figure 3.1.2

This should be at least a relatively cheap design since the region is limited by the extreme conditions. However,if $B - L$ increases, the angle $\theta$ will also increase and it will be more and more difficult for the driver to turn. Therefore, we also consider another design which can reduce the angle $\theta$ by adding a rectangular region with a length of l between the two partial-annulus regions. It will make it easier for the drivers to turn. Although it results in an extra area, it also reduces the area of the annuluses. Hence we cannot easily judge which area is larger without calculation.(see Figure 3.1.2)

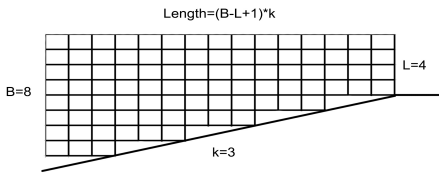
We will calculate the area of the two situations and determine which one is better at saving money in the following testing section.

### 3.0.2 Model 2

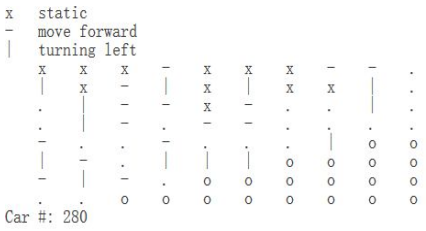
In this model, we employ a computer program to estimate the throughput in different situations. We will make the following simplifying assumptions with explanations:

- **The shape of the fan-in area is a trapezoid.** This is the usual shape of the toll plaza in our real life. We use a 2-D array to represent this simplified model. This will provide some hints for the construction of model 3 and reveal a lot of important facts that can be applied in any situation.

- **All cars travel with the same speed.** In this model, we set the velocity of different cars to be the same.
- **A car can only move a step forward, a step to the left front, or remain at rest the next moment.** Moving to the left front means trying to turn into the inner lanes. We assume so because the cars need to go back to the inner lanes from outer tollbooths.
- **All cars (except for some at special positions) can freely choose to turn left, move forward or stay at rest.**
- **If a car can go, it will never stop.**
- **Cars will never want to go to a place where there is no road.**




(c) Figure 3.2(1)




(d) Figure 3.2(2)

The fan-out area will be represented by a 2-D array. Each cell shows the state of a part of the area at a specific time. Our notations for road condition are as follows:

- " o " means the border of the fan-in region.
- " . " means there is no car at present.
- " x " means there is a car not planning to move the next moment.
- " | " means there is a car planing to go forward the next moment.
- " - " means there is a car planning turn left the next moment.

Note that there might be conflicts between a car planning to turn left with a car planing to go forward when they are adjacent. We set three different consequences if a conflict occurs .

In our model, the following parameters are involved:

- $B$ and $L$ have the same meaning as they do in the problem.
- The slope of the road is represented by a constant **k**, which is always larger than 1 (since only in this situation can cars at the bottom move), and can be seen as the inverse of the slope of the road
- **P-hesitate** denotes the average probability of a hesitation(stay at rest the next moment when there is a conflict).
- **P-moveforward** denotes the average probability of a car that wants to move forward succeeds in doing so in a conflict situation.
- **P-turnleft** denotes the average probability of a car that wants to turn left succeeds in doing so in a conflict situation.
- **Original-Cars** denotes the number of cars on the road when the simulation begins.
- **Traffic** denotes the probability of a new car showing up at each of the entrance. The higher the value, the worse the traffic.

Note that we have

$$\textbf{P-hesitate} + \textbf{P-moveforward} + \textbf{P-turnleft} = \textbf{1}.$$

Here is some information about the algorithm of our program(Using C++ language):

At the beginning, we define many function prototypes and an array "traffic" to represent the fan-in region. Then, we initialize the value of the parameters.A concise description of the functions are as follows:

$Starttraffic()$ gives an original state for the fan-in area

$displaytraffic()$ will display the array "traffic" on the screen.

$refreshtraffic()$ will let the cars move to their desired positions according to their state if they are allowed to do so. The state of a car means that it tends to go straight, turn left or stop in the coming future. If a car is not allowed to go, the function will decide whether there is a conflict between two cars or the car should just stay.

$newstate()$ will give each car a new state according to its position after moved by $refreshtraffic()$.

The function $newfirstline()$ will generate new cars on the first column of the array and give them their initial state.

By running the program many times, it provides us with a lot of useful implications.

**Some explanations**:

There are three different cases for each element of the array "traffic". It can be a car, a spare position or an occupied position. The last one means that there is no road and is on or out of the boarder of the plaza.

Cars can only turn left or go straight in the model, so when one car tend to move forward and the one on its right-hand-side wants to turn left, there is a conflict. We use the three possibilities to determine the result of the conflict, the "P-move-forward", the "P-turn-left" and the "P-hesitate", whose meanings are shown above.

If a car is originally in position (i, j) and it succeeded in turning left, then it will go to position (i-1, j+1). If it succeeded in going straight, it will go to position (i, j+1).

### 3.0.3 Model 3

**Improvements on Model 2:**

Due to the unconsidered factors, which may influence the throughput of the road, such as the proportion of Electronic Toll Collection lanes, the shape of fan-in road and the acceleration of vehicles in model 2, we construct the third model using MATLAB.

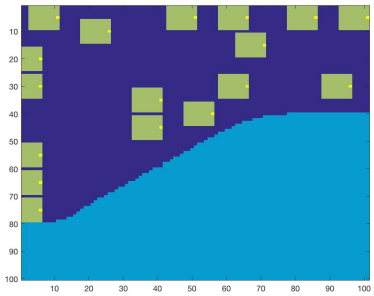



Figure 3.3,1

**Algorithm:**

Here are some information about the algorithm of model 3 (using MATLAB language): Main Process: First, we generate a fan-in area where the amount of lane declines from eight to four after the toll plaza. Two leftmost

lanes are set as ETC lanes where cars have initial velocities of 2 because they don't have to stop, and the cars from the other lanes gain initial velocities of 1. Different types of cars appear at start points with different frequencies: 1 car per 3 unit time for ETC lanes, 1 car per 5 unit time for self-driving cars on normal lanes and 1 car per 8 unit time for regular car on normal lanes. Basic traffic principles, including moving inside the road and slowing down if too close to the car in front of it, will be followed. After each move of a car, it has changeable probability to decline the velocity in order to simulate true conditions. Similar to model 2, a car possibly move either forwards or left if there's no car on the left side and in front of it. Furthermore, the car which want to turn left should let those who want move forwards go first. After a time interval set by users, the main function will display the throughput during the time interval. Several functions are defined in this model. Here are concise explanations of all the functions called in the main function: $drawBarrier(L, length)$ draw the shape of the road on an array "$L$". In this model, we adopt the barrier shape analyzed in model 1, which is composed of two arcs and one line segment. User can modify the shape of the right-hand side of the road by change the value of "length" the length of line segment; $displayCar(size\_of\_L, L, size\_of\_cars)$ display "$cars$" in the array "$L$"; $carDisappear(L, x - coordinate, y - coordinate, size\_of\_cars)$ remove the trail of the car after it make a move.

**Value assignment to accident prevention.**

We now assign a value to the accident prevention so that we can determine how likely accidents can happen in each case. We define the parameter as follows:

1. For each tollbooth, we define an "optimal path" for cars to move along;
2. For each "optimal path", we assign a value to it called $P - accident$;
3. For each "optimal path", decide all "optimal paths" generated by other tollbooths that overlap the original one;
4. For each pair of overlapping "optimal paths", we assign a number $p$ to their common area by letting $p$ equal to the product of the $P - accident$ of the two paths.
5. Integrate $p$ on all pairs of overlapping "optimal paths" and the result is denoted as $P$. Now we assign $P$ to accident prevention.

Here are some discussions about the value assignment:

- the "optimal path" consists of a line joining two arcs, where the radii of both is equal to the minimum turning radius of all cars(By assumption this is well-defined).

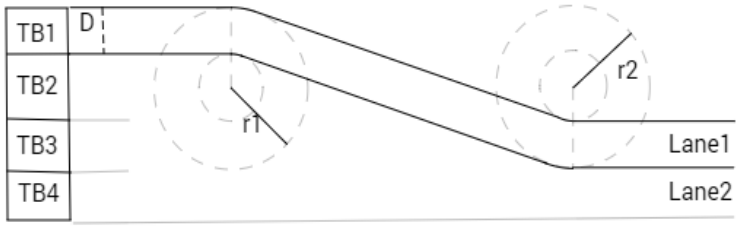



Figure 3.3.2: *this shows an "optimal path" joining TB1(tollbooth 1) and Lane1. Note that* $r1 = r2$ *is the minimum turning radius.*

- the value $P - accident$ of a path varies with the kind of tollbooth it originates from. The value is set as 4 for ETC tollbooths, and 1 for others. Here is the reason for this number. First, the average speed of a car coming out of an ETC tollbooth is approximately twice that of cars from other kinds of tollbooths. Second, suppose all cars have the same deceleration, then the stopping distance is proportional to the square of its speed.
- We have a simplifying estimation to evaluate the final integration.

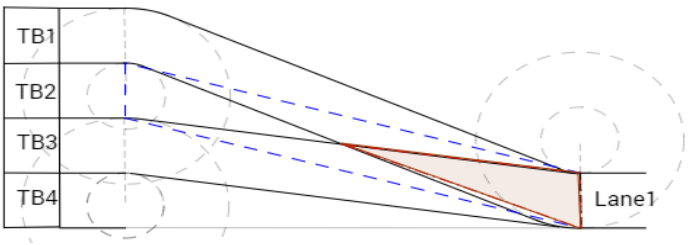



Figure 3.3.3: *this shows two "optimal paths" joining TB1, TB3 respectively with Lane1. Note that there may be more lanes below, which makes no difference to this model.*

Note that the shadowed red triangle is a pretty good approximation of the overlapping region, although a little bit smaller. The area of the triangle is half that of the blue parallelogram. In fact, using elementary knowledge of similar triangles, the area of the overlapping region of two "optimal paths" only depends on which tollbooths the paths originate from. To be precise, the area is

$$\frac{1}{n}S$$

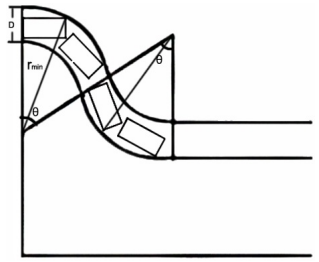
,where $n$ is the number of tollbooths between the initial two, and $S$ is the area of the parallelogram. Hence the integration becomes very easy to calculate.

- The smaller the value $P$, the better the accident prevention.

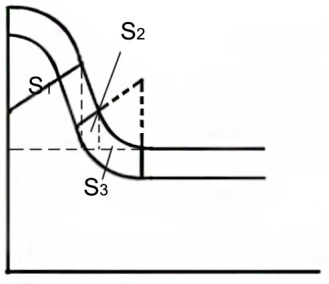
# 4 Testing the Model

## 4.1 Model 1

By the assumptions and descriptions in subsection 4.0.1, the path of the car in the first situation will be two partial-annulus regions linked smoothly, both limited by two concentric circles with radii $r_{min}$ and $r_{min} - D$, as is shown in **Figure 4.1.1**.




(e) Figure 4.1.1




(f) Figure 4.1.2

By some calculation, we can find the relationship between $\theta$ and $(B-L)$:

$$B - L = \frac{(2r - D)}{D}(1 - \cos\theta)$$

Then, we can determine the area occupied in this situation by

$$\begin{aligned} S_1 &= \frac{1}{2}\theta r^2 + \frac{1}{2}r^2 \sin\theta\cos\theta + r\sin\theta((B-L)D - r) \\ S_3 &= \frac{1}{2}(r-D)^2(\sin\theta(2-\cos\theta) - \theta) \\ S &= S_1 + S_3 \end{aligned}$$

For the second situation, as shown in Figure 4.1.2, we can also get a similar equation of $B - L$ and $\theta$.

$$B - L = \frac{l\sin\theta + (1-\cos\theta)(2r-D)}{D}$$

And the total area now is:

$$\begin{aligned} S_2 &= ((B-L)D - r)l\cos\theta + rl\cos^2\theta - \frac{l^2}{2}\sin\theta\cos\theta \\ S &= S_1 + S_2 + S_3 \end{aligned}$$

where $S_1$ and $S_3$ are the same as in the previous model

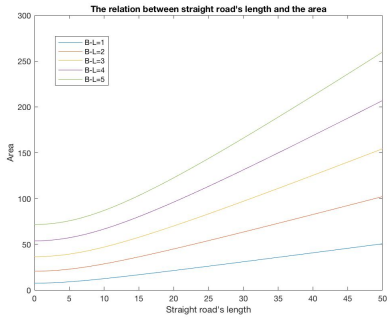



Figure 4.1.3

By analyzing the graph of this function, we conclude that S gets its minimum when $l = 0$

Therefore, the first model would be the minimum area possible. However, we cannot assume that this is the design with minimum area since there are other situations where multiple partial-annulus regions and multiple rectangles form a lane as in Figure 4.1.1. Therefore we cannot conclude that it's not possible to have smaller areas due to the limit of time. However, we can see

that when the number of rectangles and partial-annulus regions increases to infinity, actually it becomes Figure 4.1.2. As a result, we boldly hypothesize that the total area of the fan-in area will decrease as the number of rectangles increases, and it will finally reach a limit, which is the area in Figure 4.1.1.

Hence, on account of physics properties and cost, we take the model in figure 3.1.1 as the best design so far.

## 4.2 Model 2

### 4.2.1 Test 2.1

We first test the model by exploring how the slope of road side affect the throughput with the following constrains(with the notations and variables we mentioned earlier):

- $B = 8$
- $P\text{-}hesitate = 0$
- $P\text{-}moveforward = 1$
- $P\text{-}turnleft = 0$
- $Original\text{-}Cars = \frac{B*L}{2}$
- $Traffic\text{-}Situation = 0.5$

Note that we are actually considering the case where all the cars are self-driving by setting the constrains above.

With the help of our program, we change $k$ and see how $Car\text{-}Passed$ changes. We conduct five trials in total with k ranging from 1 to 5, each of 2000 unit time long. The results are in **Table 4.2.1** (see Appendix) and **Figure 4.2.1**.

| k | Average |
|---|---|
| 1 | 4420.4 |
| 2 | 3991.4 |
| 3 | 3913.4 |
| 4 | 3860.8 |
| 5 | 3788.4 |

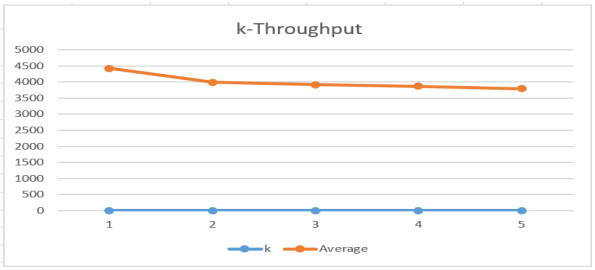



Figure 4.2.1

**Conclusion:**
$Car\text{-}Passed$ decreases with increasing $k$.
**Explanation:**
Under our assumptions, the throughput decrease when k increases, which means that less cars can merge into the lanes when the fan-in region becomes "longer". This can be explained as a "longer" region would increase the frequency of a car that wants to turn left being blocked by cars moving forward. However, it is worth noticing that we set $P\text{-}movef orward = 1$ in this experiment, which means that all cars are autonomous. Therefore we want see what happens if there are human beings driving their own cars in the following experiment.

### 4.2.2 Test 2.2

In this test, we explore how the proportion of autonomous cars influence the throughput. Using our notations and assumptions, we make the following constrains:

- $B = 8$
- $L = 4$
- $P\text{-}hesitate = 0.2$
- $k = 2$
- $Original\text{-}Cars = \frac{B*L}{2}$
- $Traffic\text{-}Situation = 0.5$

Note that this time $k = 2$ is fixed. Moreover, as $P\text{-}moveforward$ changes, $P\text{-}turnleft = 1 - P\text{-}moveforward - P\text{-}hesitate$ also changes, which means different proportion of self-driving cars.

We design five trials, with $P\text{-}moveforward$ equal to 1, 0.7, 0.5, 0.3, 0.1 and conduct the last four(because the first one, here denoted by ::1, is exactly what we have done in last test), each of 2000 unit time long.

The results are in **Table 4.2.2** (see **Appendix**).

We also present the results in **Figure 4.2.2** so as to make it clearer.

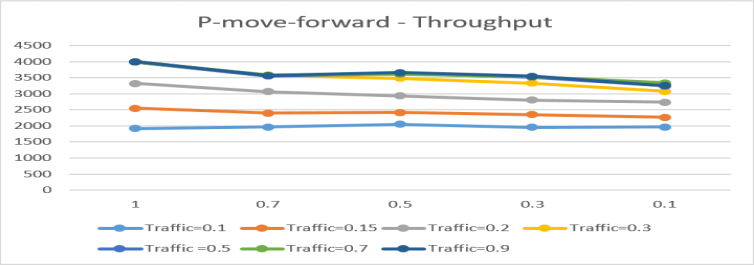



Figure 4.2.2

**Conclusion:**

- Generally speaking, the less the autonomous cars, the less the throughput.
- The less the traffic inflow, the less the throughput.
- When the traffic is light, if there are human drivers, then a moderate proportion of autonomous cars will maximize the amount of throughput

**Explanation:**

Upon direct observation, we can draw a rough conclusion that the more autonomous cars there are, the more cars can pass the point where the toll plaza and the outgoing traffic lanes merge. This can be easily understood since autonomous cars always follow the rules and rarely cause troubles. However, we also notice that when P-move-forward equals 0.5, there is a relative large amount of throughput, especially when the traffic inflow is small.

We assumed the "Turning-left goes after Moving- forward (TAM)" rule in our simulation process: If there are two cars $C1$ and $C2$ in positions (i, j)

and (i+1, j) respectively, and $C1$ wants to move forward while $C2$ needs to turn left, then $C1$ always goes and $C2$ always stays.

As the percentage of self-driving cars decreases, more cars will choose to violate the TAM rule. Although this kind of behavior causes more traffic jams, which can be explained by the generally decreasing throughput, sometimes it will also accelerate the flow of traffic by allowing more cars that are originally on the right hand side to turn into the roads on the left hand side. This change, however, results in less congestion since it is cars on the right side that frequently need to change their lanes. By reducing the number of these cars, congestion are reduced and more cars can pass the merging point. This phenomenon has a less effect on the final result when the traffic inflow is very large.

We also examine what happens if traffic inflow changes, and this is accomplished via changing the variable "Traffic" from 0.1 to 0.9. The results are shown in **Table 4.2.2(2)**(see **Appendix**) and **Figure 4.2.2(2)**, and are totally as expected. Generally speaking, when traffic inflow increases, the throughput will also increase. However, due to the limitation of the roads and the merging patterns, the throughput can never increase permanently and always reaches a limit. Besides, the two figures also provides a better demonstration for the "Decrease, then increase, and finally decrease" phenomenon in **Table 4.2.2**.

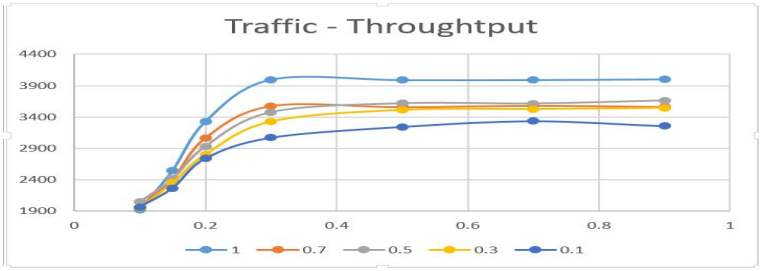



Figure 4.2.2(2)

### 4.2.3 Test 2.3

In this model, we explore how the difference between the number of tollbooths and that of lanes affect the throughput. We will see how *Car-Passed* is affected by $B - L$. Therefore, we have the following constrains:

- $L = 4$
- $P\text{-}hesitate = 0$
- $P\text{-}moveforward = 1$
- $P\text{-}turnleft = 0$
- $Original\text{-}Cars = \frac{B*L}{2}$
- $k = 2$

Note that in order to make traffic constant, the probability of a car coming out of certain tollbooth is **scaled by** $8/B$.

Again, we conduct 5 trials in total with $B - L$ ranging from 1 to 5 and $Traffic\text{-}Situation$equals to 0.1, 0.15, 0.2, 0.3, 0.5 respectively, each of 2000 unit time long.The results are in **Table 4.2.3**(see **appendix**).

We also present the results in **Figure 4.2.3** so as to make it clearer.

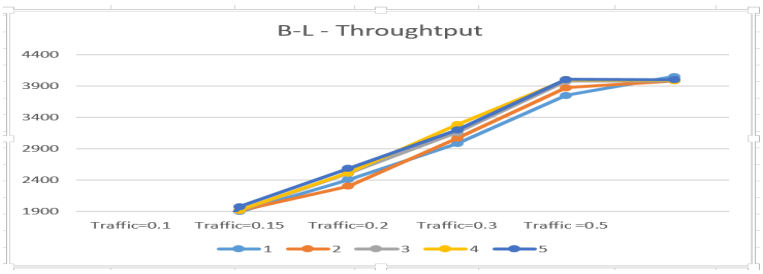



Figure 4.2.3

**Conclusion:**

- Generally, with fixed $Traffic\text{-}Situation$, $Car\text{-}Passed$ increases as $B - L$ increases.
- If the traffic is heavy, then there is no change in throughput even if there are more tollbooths.

**Explanation:**

Throughput should be larger when the number of tollbooths and the traffic situation matches. Therefore, when traffic is not so heavy, throughput increases as more cars come in. However, the more the tollbooths, the harder

for cars to turn back to their original lanes. This is because the original lines are fixed, so there is a limit for the throughput. When more tollbooths are built, the throughput reaches its limit faster.

## 4.3 Model 3

### 4.3.1 Testing Results

After running the model 3 program many times, we actually get similar results as in model 2, which means that all the conclusions drawn in the second model can be generally applied to many situations. Therefore, we don't give a detailed description about the data we collect. Instead, we mention some differences and similarities in the results of the two models. All the data collected for model 3 are attached in the appendix.

First of all, we discover when the length of the rectangle increases, (the angle $\theta$ declines) the throughput declines. Therefore, in order to maximize throughput, we should also let $l = 0$ and set the shape of the plaza to be the same as the first design in model 1. This is an intriguing result since the cost of the plaza is minimized also when $l = 0$, which reveals the fact that the first design is a great design that can satisfy at least two of our requirements.

For the other conclusions in model 2, most of them are also valid after the design changes its shape. In particular, when the number of autonomous cars increases, we can find that throughput will still increase, but the "interesting fact" in model 2 in this situation is less obvious. The "interesting fact" refers to the phenomenon that if there is a certain proportion of people violating the traffic rules when the traffic is light, it would actually increase the throughput.

Also, we discover the same fact that lanes always have their limits and an increase in tollbooth numbers cannot increase the throughput amount permanently since the number of lanes is fixed. This limitation shows that the number of tollbooths cannot greatly affect the Throughput.

Another result we get is that the automated tollbooths will not have great impact on the throughput. However, the number and positions of electronic collecting booths will result in different throughput since the velocity of cars coming out of electronic collecting booths is much faster. Generally speaking, the more the electronic collecting booths, the more throughput. The positions of the electronic collecting booths will be discussed later

### 4.3.2 Accident Prevention And Merging Pattern

With results from [1], we know that with the proportion of cars equipped with ETC transponder changes, the design of tollbooths changes. For example, when the proportion is low, ETC tollbooths should be on the outermost side of the station. With our definition of accident prevention, we may take the case where $B = 4$ and $L = 2$ as an example to see how to determine the merging pattern.

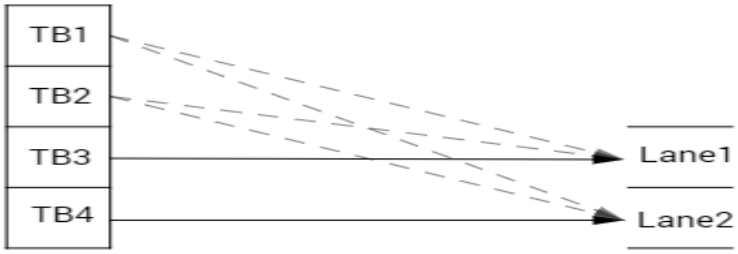



Figure 4.3.1: $TB1$ is an ETC tollbooth

As shown in **Figure 4.3.1**, cars from $TB3$ and $TB4$ will choose to move forward. However, cars from $TB1$ and $TB2$ may choose to go through either $Lane1$ or $Lane2$. Note that $TB1$ is an ETC tollbooth, $P - accident$ is 4 there.

We calculate the integration $P$ of the different cases here:

| P | TB2-L1 | TB2-L2 |
|---|---|---|
| TB1-L1 | 7 | $\frac{7}{2}$ |
| TB1-L2 | $\frac{19}{3}$ | $\frac{41}{6}$ |

**Clarification:**

We see that when cars from TB1 go to Lane1 and cars from TB2 go to Lane2, the value of P is the smallest, which means the best accident prevention. This is easily explained as follows:

Cars on ETC lanes are more dangerous. In order to strengthen accident prevention, fewer "optimal paths" should overlap with them. Hence ETC lanes should point to outermost lanes after the fan-in region and other lanes should avoid overlapping with them as much as possible.

# 5 Results

Here is our optimal solution to the problem:

1. The outermost side of the plaza is a combination of two circles with radii equal to the minimum turning radius of the cars.
2. ETC tollbooths should be on the outermost side.
3. Cars from the ETC tollbooths should move to the outermost lane after the plaza.
4. Cars from other kinds of tollbooths should either go straightforward(if possible) or go to the outermost lane other than those for cars from ETC tollbooths.

# 6 Sensitivity Test and Stability Test

We examined the effect of changing variables first. We tried different value of $B - L$, *P-hesitate*, *P-moveforward*, *P-turnleft*, *Original-Cars*, $Traffic$-*Situation* and $k$, none of which has a determining effect of the optimal solution.

However, $Traffic$-*Situation* has an enormous effect on the throughput. Even when $Traffic$-*Situation* gets worse a bit, the throughput almost reaches its limit.

Hence our model is very reliable with respect to sensitivity test, but not so stable.

# 7 Strengths and Weaknesses

## 7.1 Strengths

- We take into account several relating factors and build our models step by step until the final one, which is much more complicated than the first two models.
- The model should be a nice simulation of real life situation since most relating subjects have been covered.

- The second model, which is a much simplified simulation model, is actually heuristic. Its results apply well in the third model, and can be easily understood.

## 7.2 Weaknesses

- All the data we collect and analyze is from our experiments. We didn't really use the data of roads and toll plazas. Therefore, there might be biased conclusions.
- In the first model, we don't have enough time to examine whether the hypothesis that the first design minimizes the area is true. This would involve a lot of mathematical calculation and analysis, and could be determined if we were given more time.
- The second model and the third model are actually similar in some situations, and it is a waste of time and energy.
- We haven't analyzed all possible outcomes by changing all the variables that we have mentioned. If we have more time, we can give a more general result.

# References

[1] Pratelli A, Schoen F. *Optimal Design of Motorway Toll Stations* PROCEEDINGS OF THE EUROPEAN TRANSPORT CONFERENCE (ETC) 2003 HELD 8-10 OCTOBER 2003, STRASBOURG, FRANCE, 2003.

[2] Danko J, Culewicz V. *Insight Through Innovation: A Dynamic Approach To Demand Based Toll Plaza Lane Staffing* Proceedings of the 26th conference on Winter simulation. Society for Computer Simulation International, 1994.

[3] Wolfram S. *Statistical mechanics of cellular automata[J].* Reviews of modern physics, 1983.

[4] Polus, A. *Methodology and simulation for toll plaza analysis.* Road & transport research. Vol. 5, no. 1. 1996.

# 8 Appendix

## 8.1 Tables for Model 2

| Other Variables | | Independent Variable | Car Passed | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | k | First trial | Second trial | Third trial | Fourth trial | Fifth trial | Average |
| B | 8 | 1 | 4407 | 4451 | 4424 | 4384 | 4436 | 4420.4 |
| L | 4 | 2 | 3984 | 3976 | 3991 | 4008 | 3998 | 3991.4 |
| P-hesitate | 0 | 3 | 3915 | 3922 | 3925 | 3903 | 3902 | 3913.4 |
| P-move-forward | 1 | 4 | 3848 | 3834 | 3881 | 3879 | 3862 | 3860.8 |
| P-turn-left | 0 | 5 | 3796 | 3747 | 3811 | 3818 | 3770 | 3788.4 |
| Original cars | B*L/2 | | | | | | | |
| Traffic | 0.5 | | | | | | | |
| | | | | | | | | |
| | | | | | | | total:2000 | |

Table 4.2.1

| Other Variables | | Independent Variable | Car Passed | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | P-move-forward | First trial | Second trial | Third trial | Fourth trial | Fifth trial | Average |
| B | 8 | ::1 | 3984 | 3976 | 3991 | 4008 | 3998 | 3991.4 |
| L | 4 | 0.7 | 3562 | 3529 | 3567 | 3536 | 3610 | 3560.8 |
| P-hesitate | 0.2 | 0.5 | 3641 | 3613 | 3627 | 3591 | 3625 | 3619.4 |
| P-turn-left | 0.8 - P-move-forward | 0.3 | 3504 | 3461 | 3569 | 3535 | 3510 | 3515.8 |
| k | 2 | 0.1 | 3203 | 3314 | 3249 | 3244 | 3221 | 3246.2 |
| Original cars | B*L/2 | | | | | | | |
| Traffic | 0.5 | | | | | | | |

Table 4.2.2

| P-move-forward | 1 | 0.7 | 0.5 | 0.3 | 0.1 |
|---|---|---|---|---|---|
| Traffic=0.1 | 1922.2 | 1962.2 | 2048.8 | 1952.22 | 1966.8 |
| Traffic=0.15 | 2546.6 | 2401.8 | 2417.2 | 2351.4 | 2267.2 |
| Traffic=0.2 | 3325.8 | 3063.6 | 2931.6 | 2802.2 | 2740.4 |
| Traffic=0.3 | 3997.6 | 3578.8 | 3477.2 | 3330.4 | 3072.8 |
| Traffic =0.5 | 3991.4 | 3560.8 | 3619.4 | 3515.8 | 3246.2 |
| Traffic=0.7 | 3992.2 | 3584.4 | 3616.2 | 3528.6 | 3336.4 |
| Traffic=0.9 | 4004.2 | 3564.4 | 3664.6 | 3548.2 | 3256.8 |

Table 4.2.2(2)

| Other Variables | | Independent Variable | Car Passed | | | | |
|---|---|---|---|---|---|---|---|
| | | B-L | Traffic=0.1 | Traffic=0.15 | Traffic=0.2 | Traffic=0.3 | Traffic =0.5 |
| L | 4 | 1 | 1901.2 | 2411.4 | 2984.6 | 3752.4 | 4051.2 |
| P-hesitate | 0 | 2 | 1916.4 | 2306 | 3067.2 | 3873.6 | 3978.4 |
| P-move-forward | 1 | 3 | 1912.8 | 2506.4 | 3155.4 | 3976.2 | 3982.4 |
| P-turn-left | 0 | 4 | 1926.4 | 2515.8 | 3283.2 | 3997.2 | 3987.2 |
| Original cars | B*L/2 | 5 | 1980.2 | 2588.6 | **3201.2** | **4002.2** | **4000.8** |
| k | 2 | | | | | | |
| | | | | | | | |
| | | | | | total:2000 | | |

*In order to make traffic constant, the probability of a car coming out of certain toll-booth is scaled by 8/B

Table 4.2.3

## 8.2 Codes for Model 2

```
#include <iostream>
#include <time.h>
#include <iomanip>
#include <math.h>
using namespace std;

/*8 is spare
* 0 is no road
* 1 is static
* 2 is move forward
* 3 is turn left
*/

const int B = 8;
const int L = 4;
const int k = 2;  //Every k steps , the number of positions will minus 1
const int Length = k * (B - L + 1);
const double p_move_forward_go_first = 0.1; //The possibillity of hesitation
const double p_turn_left_go_first = 0.7;
const double p_hesitate = 1 - p_move_forward_go_first - p_turn_left_go_first;
const int start = B*L / 2;//The starting number of cars on the road
const double out_car_probability = 0.5* double(8)/B;
double danger = 0;     //How many accidents happened
                       //The accident rate is now set to 0.001
```

```
int traffic[B][Length];
//Forward, left, hesitate are three possible outcomes

//|__________|__________|________|
//  Forward     left       hesitate


// prototypes


void displaytraffic();
void starttraffic();
void refreshtraffic();
void newstate();
void newfirstline();

int main() {
double danger = 0;
starttraffic();
displaytraffic();
int m, cnt = 0;
do {
cin >> m;
for (int i = 0; i < L; i++) {
if (traffic[i][Length-1] == 2)
cnt = cnt + 1;
}
refreshtraffic();
displaytraffic();
cout << "Car #: " <<cnt << endl;
} while (m);
cout << danger;
return 0;
}

void displaytraffic() {
cout << "x   static" << endl;
cout << "-   move forward" << endl;
```

```
cout << "|   turning left" << endl;
char displaymatrix[B][Length];
for (int i = 0; i < B; i++) {
for (int j = 0; j < Length; j++) {
switch (traffic[i][j]) {
case(0):displaymatrix[i][j] = 'o';
break;
case(1):displaymatrix[i][j] = 'x';  //How you display the matrix
break;
case(2):displaymatrix[i][j] = '-';
break;
case(3):displaymatrix[i][j] = '|';
break;
case(8):displaymatrix[i][j] = '.';
break;

}
}
}

for (int i = 0; i < B; i++) {
for (int j = 0; j < Length; j++) {
cout << setw(5) << displaymatrix[i][j];
}
cout << endl;
}
}//1 is static, 2 is move forward, 3 is turning left, 8 is nothing
void starttraffic() {
srand(time(0));
for (int j = 0; j < Length - 1; j++)//the first row can only stay or move forward
traffic[0][j] = rand() % 2 + 1;
traffic[0][Length - 1] = 2;
for (int i = 1; i < L; i++) {     //after the first row,3 possibilities
for (int j = 0; j < Length - 1; j++) {
traffic[i][j] = rand() % 3 + 1;
}
traffic[i][Length - 1] = 2;   //The last column can only move forward
}
```

```
for (int i = L; i < B; i++) {     //after the 4th line
for (int j = 0; j < Length; j++) {
if (j < (Length - k*(i - L + 1))) {
traffic[i][j] = rand() % 3 + 1;
}
else {      //The "not road situation"
traffic[i][j] = 0;
}
}
}
for (int i = L; i < B; i++) {
if (traffic[i][Length - k * (i - L + 1) - 1] == 2) {
//if not allowed to go forward
if (rand() % 2 == 0) {
traffic[i][Length - k * (i - L + 1) - 1] = 1;
//either stop or turn left
}
else {
traffic[i][Length - k * (i - L + 1) - 1] = 3;
}
}
}     //the one on the boarder can only want to move forward or turn left
for (int k = B - 1; k >= 1; k--) {
if (traffic[k][0] == 1) {
traffic[k][0] = rand() % 2 + 2;
}
traffic[0][0] = 2;
}
if (::k == 1 && traffic[B - 1][0] == 2) {
traffic[B - 1][0] = 3;
}
//Randomly generate 8
for (int k = 0; k < start; k++) {
int row = rand() % B;
int column = rand() % Length;
if (traffic[row][column] != 0) {
traffic[row][column] = 8;
}
```

```
}
}//Refresh traffic status
void refreshtraffic() {
for (int k = L - 1; k >= 0; k--) {
traffic[k][Length - 1] = 8;
//The last line must be updated first
}
for (int j = Length - 2; j >= 0; j--) {
for (int i = B - 1; i >= 0; i--) {
//For the last column, cannot do this

if (traffic[i][j] == 2) {
if (traffic[i][j + 1] != 1) {
//if the front is not hindered, then go.
traffic[i][j + 1] = 2;
traffic[i][j] = 8;
}
else {
traffic[i][j] = 1;
}
}
else if (traffic[i][j] == 3) {
if (traffic[i - 1][j] != 2) {
//if no car on the left is going forward
if (traffic[i - 1][j + 1] != 1) {
traffic[i - 1][j + 1] = 3;
traffic[i][j] = 8;
}
else
traffic[i][j] = 1;
}
//if the left side wants to go forward
else {
double k = double(rand() % 10) / 10;
if (k < p_move_forward_go_first) {
traffic[i][j] = 1;
}
else if (k > 1 - p_hesitate) {
```

```
traffic[i][j] = 1;  //if hesitate
traffic[i-1][j] = 1;
}
else {
danger = danger + 0.001;
traffic[i - 1][j + 1] = 3;//turn left first
traffic[i][j] = 8;//update
traffic[i-1][j] = 1;
}
}
}
}
}
newfirstline();
    newstate();
}
void newstate() {
//update the cars except for the first line and the first row
for (int i = 1; i < B; i++) {
for (int j = 1; j < Length - 1; j++) {
if (traffic[i][j] != 0 && traffic[i][j] != 8) {
if (traffic[i][j + 1] == 8) {              //If no car, must go
if (traffic[i - 1][j + 1] != 8) {
traffic[i][j] = 2;
}
else {
traffic[i][j] = rand() % 2 + 2;
}
}
else if (traffic[i - 1][j + 1] == 8) {
//if there is a car in the front
traffic[i][j] = 3;
}
else {
traffic[i][j] = rand() % 3 + 1;
}
}
}
```

```
}
//update the first row, including the first one
for (int j = 0; j < Length; j++) {
//The ones on the first row
if (traffic[0][j] != 8) {
if (traffic[0][j + 1] == 8) {
traffic[0][j] = 2;
}
else {
traffic[0][j] = rand() % 2 + 1;
}
}
}
//update the one on the boarders
for (int i = L; i < B; i++) {
if (traffic[i][Length - k * (i - L + 1) - 1] == 2) {
//if not allowed to go forward
if (rand() % 2 == 0) {
traffic[i][Length - k * (i - L + 1) - 1] = 1;
//either stop or turn left
}
else {
traffic[i][Length - k * (i - L + 1) - 1] = 3;
}
}
}
//update the last column
for (int i = 0; i < L; i++) {   //update the last column
if (traffic[i][Length - 1] == 3 || traffic[i][Length - 1] == 1) {
traffic[i][Length - 1] = 2;
}
}
//update the first column, when =1 (except for the first one)
for (int i = 1; i < B; i++) {
if (traffic[i][0] == 1) {
//other cases have been covered by the newfirstline function
if (traffic[i + 1][0] == 8) {//If no car, must go
if (traffic[i - 1][1] != 8) {
```

```
traffic[i][0] = 2;
}
else {
traffic[i][0] = rand() % 2 + 2;
}
}
else if (traffic[i - 1][1] == 8) {
//if there is a car in the front
traffic[i][0] = 3;
}
else {
traffic[i][0] = rand() % 3 + 1;
}
}
if (::k == 1 && traffic[B - 1][0] == 2) {
traffic[B - 1][0] = 3;
}
}
}

void newfirstline() {
//if there is no car, must be cleared first

for (int k = 0; k < B - 1; k++) {
if (traffic[k][0] != 1) {
traffic[k][0] = 8;
}
}
for (int k = 0; k < B; k++) {
double t = double(rand() % 1000 + 1)/ 1000;
if (t <= out_car_probability) {
if (traffic[k][0] == 8) {
traffic[k][0] = rand() % 2 + 2;
//new car can only go straight or turn left
}
}
if (::k == 1 && traffic[B - 1][0] == 2) {
traffic[B - 1][0] = 3;
```

```
}
if (traffic[0][0] == 3) {
traffic[0][0] = 2;
}
}

}
```

## 8.3 Tables for Model 3

| I | Average |
|---|---|
| 0 | 1153.2 |
| 10 | 1103.4 |
| 20 | 1087.4 |
| 30 | 1016.4 |
| 40 | 998.2 |

Figure 8.3.1

| P-move-forward | 1 | 0.7 | 0.5 | 0.3 | 0.1 |
|---|---|---|---|---|---|
| Traffic=0.1 | 660 | 633 | 620 | 617 | 613 |
| Traffic=0.15 | 745 | 733 | 716 | 715 | 689 |
| Traffic=0.2 | 940 | 873 | 842 | 833 | 794 |
| Traffic=0.3 | 1006 | 1004 | 978 | 961 | 822 |
| Traffic =0.5 | 1101 | 999 | 1020 | 988 | 921 |
| Traffic=0.7 | 1107 | 1000 | 1013 | 991 | 924 |
| Traffic=0.9 | 1086 | 1004 | 1025 | 991 | 933 |

Figure 8.3.2

| B-L | Traffic=0.1 | Traffic=0.15 | Traffic=0.2 | Traffic=0.3 | Traffic =0.5 |
|---|---|---|---|---|---|
| 1 | 612 | 674 | 786 | 997 | 1006 |
| 2 | 627 | 699 | 894 | 1002 | 1010 |
| 3 | 634 | 712 | 933 | 1004 | 1016 |
| 4 | 656 | 755 | 1000 | 1011 | 1027 |
| 5 | 689 | 803 | 1019 | 1020 | 1028 |

Figure 8.3.3

## 8.4 Codes for Model 3

```
function car_num = trail2 (layer)
%this function draw a circle
%   Detailed explanation goes here

M = 100;
L = zeros(M,M+1); %create the grid
V = L; %Create a velosity matrix

per = 10;

L = drawBarrier(L,50);

wide = 79; num = 10; vmax = 2; roads = 8; size = 9;
startpoint = round(roads .* rand(1, num) + 0.5);
startpoint = startpoint .* 10 - 5;
L(startpoint,1) = 3;   %generate the cars
count = L(5:10:wide,1)' ./ 3; %count the time
L = displayCar(M,L,size);
car_num = 0; %count the number of cars which go through the road
for n = 1:roads
    if count(n) == 0
        count(n) = round(per*rand(1,1)-(per-0.5));
        %control the frequency of cars' appearance.
    end
```

```
end
A = ones(1,roads);  %creat an array to be the denominator
for t = 1:layer
    for x=1:wide    %give the car velocities
         if L(x,1) == 3 && x <= 0.25*wide
            V(x,1) = 1;   %for etc cars
        else
            V(x,1) = 0;  %for normal cars
        end
    end
    L = displayCar(M,L,size);
    for y = M+1:-1:1
        for x = 1:M
            if L(x,y) == 3 %find the car
                if L(x-4:x+4,min(M+1,y+1):min(M+1,y+9)) == 0
%there's nothing in front of it excluding the car at the last line
                    if y == M-4 && (V(x,y) == 1 || V(x,y) == 2)
                    % the car is at the last 2 line and v = 2
                        L = carDisappear(L,x,y,size);
                        V(x,y) = 0;
                        car_num = car_num + 1;
% no barrier in front of it and not at the last 2 line
                elseif L(max(1,x-6),max(1,y-10):min(M+1,y+6)) == 0
                %there's no car on the left
                                    ran = rand(1,1);
                                    if ran < 0.5 %go straight
                                            V(x,y) = 0;
                                        end
                                        L(x,y+min(V(x,y)+1,vmax)*5) = 3;
          V(x,y+min(V(x,y)+1,vmax)*5) = min(V(x,y)+1,vmax); %make a move
                                        ran1 = rand(1,1);
                                        if ran1 >= 0.8
    V(x,y+min(V(x,y)+1,vmax)*5) = max(V(x,y+min(V(x,y)+1,vmax)*5) - 2,0);
                                        end
                                        if ran1 < 0.8 && ran1 >= 0.5
    V(x,y+min(V(x,y)+1,vmax)*5) = max(V(x,y+min(V(x,y)+1,vmax)*5) - 1,-1);
                                        end
                                        if V(x,y) ~= -1
```

```
                                        L = carDisappear(L,x,y,size);
                                        V(x,y) = 0;
                                    end
                                    L = displayCar(M,L,size);
                                else %turn left
                                    L(x-(size+1)/2,y+5) = 3;
                                    V(x-(size+1)/2,y+5) = V(x,y);
                                    L = carDisappear(L,x,y,size);
                                    V(x,y) = 0;
                                    L = displayCar(M,L,size);
                                end
                            else if L(x-4:x+4,min(M+1,y+1):min(M+1,y+12)) ~= 0
                                    V(x,y) = 0;
                                end
                            L(x,y+min(V(x,y)+1,vmax)*5) = 3;
                            V(x,y+min(V(x,y)+1,vmax)*5) = min(V(x,y)+1,vmax);
                                ran1 = rand(1,1);
                                if ran1 >= 0.8
   V(x,y+min(V(x,y)+1,vmax)*5) = max(V(x,y+min(V(x,y)+1,vmax)*5) - 2,0);
                                end
                                if ran1 < 0.8 && ran1 >= 0.4
V(x,y+min(V(x,y)+1,vmax)*5) = max(V(x,y+min(V(x,y)+1,vmax)*5)-1,-1);
                                end
                                if V(x,y) ~= -1
                                   V(x,y) = 0;
                                   L = carDisappear(L,x,y,size);
                                end
                                L = displayCar(M,L,size);
                            end
                        end
                    else if y == M+1
          %there's sth in front of the car or the car is at the last line
                            L = carDisappear(L,x,y,size);
                            V(x,y) = 0;
                            car_num = car_num + 1;
                        else
                                if L(max(1,x-6),max(1,y-9):min(M+1,y+5)) == zeros
                                %there's no car on the left
```

```
                                if L(x-4:x-1,y:min(M+1,y+7)) == zeros
                                   L(x-5,y+5) = 3;
                                   L = carDisappear(L,x,y,size);
                                   V(x-5,y+5) = V(x,y);
                                   V(x,y) = 0;
                                   L = displayCar(M,L,size);
                                else
                                    L(x-5,y) = 3;
                                   L = carDisappear(L,x,y,size);
                                   V(x-5,y) = V(x,y);
                                   V(x,y) = 0;
                                   L = displayCar(M,L,size);
                                end
                            else
                                    % some car on the left then stop.
                                V(x,y) = 0;

                            end

                        end
                    end
                end
            end
        end
    end
    count = count + 1;
    Remain = rem(count,per*A);
    L = displayCar(M,L,size);
    for x = 1:roads
        if Remain(x) == 0 && L(10*x-6,1) == 0 && L(10*x-4,1) == 0
            L(10*x-5,1) = 3;
            if x <= 0.25*roads
                V(10*x-5,1) = 1;
            else
                V(10*x-5,1) = 0;
            end
        else if Remain(x) == 0 && (L(10*x-6,1) ~= 0 || L(10*x-4,1) ~= 0)
                count(x) = count(x) - 1;
            end
```

```
        end
    end
    L = displayCar(M,L,size);
    imagesc(L);
    pause(0.01);
end
end


function N = displayCar (M,L,size)
%this function display the car on the matrix
for y = M+1:-1:1
    for x = 1:M
        if L(x,y) == 3
            for a = x-(size-1)/2:x+(size-1)/2
                for b = y:-1:max(1,y-size+1)
                    L(a,b) = 2;
                end
            end
        L(x,y) = 3;
        end
    end
end
N = L;
end

function D = carDisappear (L,x,y,size)
%this function disappear a car
for a = x-(size-1)/2:x+(size-1)/2
      for b = y:-1:max(1,y-size+1)
             L(a,b) = 0;
      end
end
D = L;
end

function B = drawBarrier (L,length)
%this function draw the barrier
```

```
r = 32.5;l = length;
theta = asin((2*length*(20-r)+2*(l^2*(r-20)^2-
40*(4*r^2+l^2)*(10-r))^0.5)/(4*r^2+l^2));
ax = 80 -r+r*cos(theta); ay = r*sin(theta);
bx = 40 + r - r*cos(theta);
by = r*sin(theta)+l*cos(theta); o1x = 80-r;
o1y = 0; o2x = 40+r; o2y = 2*r*sin(theta)+l*cos(theta);

for y = 1:ay
    for x = ax:100
        if ((x-o1x)^2 + (y-o1y)^2) >= r^2
            L(ceil(x),ceil(y)) = 1;
        end
    end
end
for y = by:o2y
    for x = 40:100
        if ((x-o2x)^2 + (y-o2y)^2) <= r^2
            L(ceil(x),ceil(y)) = 1;
        end
    end
end
for y = ay:by
    for x = bx:100
        if (y-by)/(x-bx)>=(by-ay)/(bx-ax)
            L(round(x),round(y)) = 1;
        end
    end
end
for y = round(o2y):101
    for x = 40:100
    L(x,y) = 1;  %straight line of right-hand side
    end
end
for y = 1:101
    for x = 80:100
        L(x,y) = 1;
    end
```

```
end
B = L;
end
```