2014

A Comparative Analysis of Different Dilemma Zone Countermeasures at Signalized Intersections based on Cellular Automaton Model

Yina Wu
University of Central Florida

A COMPARATIVE ANALYSIS OF DIFFERENT DILEMMA ZONE COUNTERMEASURES AT SIGNALIZED INTERSECTIONS BASED ON CELLULAR AUTOMATON MODEL

by

YINA WU
B.S. Beijing Jiaotong University， 2012

A dissertation submitted in partial fulfillment of requirements
for the degree of Master of Science
in the Department of Civil and Environmental Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2014

Major Professor: Mohamed Abdel-Aty

ABSTRACT

In the United States, intersections are among the most frequent locations for crashes. One of the major problems at signalized intersection is the dilemma zone, which is caused by false driver behavior during the yellow interval. This research evaluated driver behavior during the yellow interval at signalized intersections and compared different dilemma zone countermeasures. The study was conducted through four stages.

First, the driver behavior during the yellow interval were collected and analyzed. Eight variables, which are related to risky situations, are considered. The impact factors of drivers' stop/go decisions and the presence of the red-light running (RLR) violations were also analyzed.

Second, based on the field data, a logistic model, which is a function of speed, distance to the stop line and the lead/follow position of the vehicle, was developed to predict drivers' stop/go decisions. Meanwhile, Cellular Automata (CA) models for the movement at the signalized intersection were developed.

In this study, four different simulation scenarios were established, including the typical intersection signal, signal with flashing green phases, the intersection with pavement marking upstream of the approach, and the intersection with a new countermeasure: adding an auxiliary flashing indication next to the pavement marking. When vehicles are approaching the intersection with a speed lower than the speed limit of the intersection approach, the auxiliary flashing yellow indication will begin flashing before the yellow phase. If the vehicle that has not passed the pavement marking before the onset of the auxiliary flashing yellow indication and can see the flashing indication, the driver should choose to stop during the yellow interval. Otherwise, the driver should choose to go at the yellow duration. The CA model was employed

to simulate the traffic flow, and the logistic model was applied as the stop/go decision rule. Dilemma situations that lead to rear-end crash risks and potential RLR risks were used to evaluate the different scenarios.

According to the simulation results, the mean and standard deviation of the speed of the traffic flow play a significant role in rear-end crash risk situations, where a lower speed and standard deviation could lead to less rear-end risk situations at the same intersection. High difference in speed are more prone to cause rear-end crashes. With Respect to the RLR violations, the RLR risk analysis showed that the mean speed of the leading vehicle has important influence on the RLR risk in the typical intersection simulation scenarios as well as intersections with the flashing green phases' simulation scenario.

Moreover, the findings indicated that the flashing green could not effectively reduce the risk probabilities. The pavement marking countermeasure had positive effects on reducing the risk probabilities if a platoon's mean speed was not under the speed used for designing the pavement marking. Otherwise, the risk probabilities for the intersection would not be reduced because of the increase in the RLR rate. The simulation results showed that the scenario with the pavement marking and an auxiliary indication countermeasure, which adds a flashing indication next to the pavement marking, had less risky situations than the other scenarios with the same speed distribution. These findings suggested the effectiveness of the pavement marking and an auxiliary indication countermeasure to reduce both rear-end collisions and RLR violations than other countermeasures.

To my parents
Hui Ma, Bin Wu

Who encourage me to realize my dream

ACKNOWLEDGMENTS

I would like to thank all three members of my thesis committee, Dr. Aty, Dr. Lee and Dr. Eluru, for their commitment to the successful completion of this research effort. In particular, Dr. Aty, who is a respectable mentor, has given me countless help and guidance on both my academic and individual development.

I would also like to express my thanks to Dr.Chen, Dr. Yan and Dr. Jia from Beijing Jiaotong University, where I spent my first year as a master's student. Dr. Chen, who is my mentor at BJTU, has imparted knowledge and guidance to me. Dr. Yan has always donated his time to provide his help and advice on both my career and individual development. Dr. Jia has provided invaluable guidance with my cellular automata model and simulation work.

Meanwhile, I appreciate the help that I have received from the members of our research group, who are always willing to share their ideas and their knowledge. I would also like to thank my friends, Jiawei Wu, Binya Zhang and Vicki League, who have given me a great deal of encouragement. I am thankful for the company of my boyfriend and all of the things he has done for me while I pursued the research project.

In addition, the strong support of my family encourages me to complete my experience as a master's student. I would like to thank my parents for their support and contribution. Without their support, I never could have flown overseas and fulfilled my dream of continuing my education in the United States.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF NOTATIONS

RLR	Red-light running
CA	Cellular Automata
PMAIC	pavement marking and an auxiliary indication countermeasure
x_n	Position of the n^{th}vehicle
v_n.	Velocity of the n^{th}vehicle
d_n	Headway between the vehicle and the vehicle in front
RS	Risky Situations
L	Road length
t	Number of time step
acc	Maximum acceleration
dec	Maximum deceleration
v_{int}	Initial speed of vehicle
BRAKE	Slam on the brake
RS1	Risk situations caused by stopped cars
RS2	Risk situations caused by non-stopped cars

CHAPTER 1 INTRODUCTION

1.1. Background

In the United States, intersections are among the most frequent locations for crashes. In 2012, 2,498,000 vehicles were involved in intersection or intersection-related crashes at signalized intersections. Of all these crashes, about 4,460 vehicles were involved in fatal crashes, and 840,000 vehicles were involved in injury crashes. Moreover, about 1,333,000 rear-end crashes occurred in 2012, representing approximately 24% of all crashes that took place in the U.S. (Traffic safety facts 2012, 2012). In term of red-light running (RLR), 683 people were killed while about 133,000 people were injured in crashes that involved red light running in 2012 ("Red light running", 2013). The engineers have estimated that at least 10 percent can be directly attributed to RLR (Sunkari et al., 2003).

At the onset of the yellow indication, drivers who are approaching the intersection must make a quick decision to either stop or cross the intersection. Among all the intersection-related crashes, yellow-phase-related crashes caused by the dilemma zones are of significant concern to transportation engineers. The dilemma zone, which is also known as the 'indecision period', describes the region which begins at the position where most people choose to stop and ends at the position where most people choose to cross the intersection at the onset of the yellow indication of the signal. The indecision period of the driver may have a negative impact on crash risks. Sometimes, RLR violations occur because of the drivers' false stop/go judgment, and rear-end crashes happen due to the different drivers' decisions at the yellow duration. Many different types of dilemma zone countermeasures have been proposed, including adding the flashing yellow phases or the pavement marking at upstream of the intersection to help drivers' make better decisions during the yellow interval.

This research simulates traffic flow at signalized intersections based on the Cellular Automata (CA) model. Scenarios were established based on different dilemma zone countermeasures. Risk situations of both rear-end crash risks and potential red-light running violations risks are estimated during the simulations.

1.2. Research Objectives

The principal objective of this research is to analyze the driver behavior during the yellow interval and conduct a comparative study of different dilemma zone countermeasures based on the CA model. The specific aims are to:

1) Analyze and model the driver behavior, especially the stop/go decisions and RLR violations, during the yellow interval at a typical intersection.
2) Simulate the driver behavior based on the CA model under different scenarios.
3) Propose a new method to improve the previous pavement marking countermeasure.
4) Conduct the intersection's high risk situations analysis of four different scenarios, which include the typical intersection signal, the intersection with the flashing green phases, the intersection with the pavement marking at the upstream of the stop line and the intersection with a new countermeasure that have both pavement marking and an auxiliary flashing yellow indication, which is referred to as " pavement marking and an auxiliary indication countermeasure (PMAIC)".

1.3. Organization of the Thesis

This thesis contains seven chapters. The first chapter is an introductory chapter including background, research objects and organization of the thesis.

Chapter 2 delves into the literature of critical issues related to the research topic. It highlights the dilemma zone and different countermeasures, and also provides the driver behavior information and countermeasures information for the scenario construction. In addition, chapter 2 also presents the basic concepts of the CA models and its development.

Chapter 3 focuses on data collection and field data analysis. Eight variables, which were extracted from the field data, were considered in this study. Meanwhile, impact factors of the stop/go decisions and the red-light running violations were analyzed.

Chapter 4 focuses on methodology research, which includes the stop/go decisions model based on the logistic regression and simulation model based on the CA model. General rules and introduction of the simulation are developed in chapter 4.

Chapter 5 describes the design of each scenario and analyzes results of different scenarios. Detailed information for each scenario is provided in this chapter. The impact of the expected mean speed and the standard deviation of leading vehicles on risky situations at different scenarios were analyzed in this chapter.

Chapter 6 conducts the comparative analysis between different scenarios. Based on the simulation results, the effect of different dilemma zone countermeasures is evaluated in this chapter.

Chapter 7 provides the conclusion and recommendations derived from the simulation and analyses presented in previous chapters.

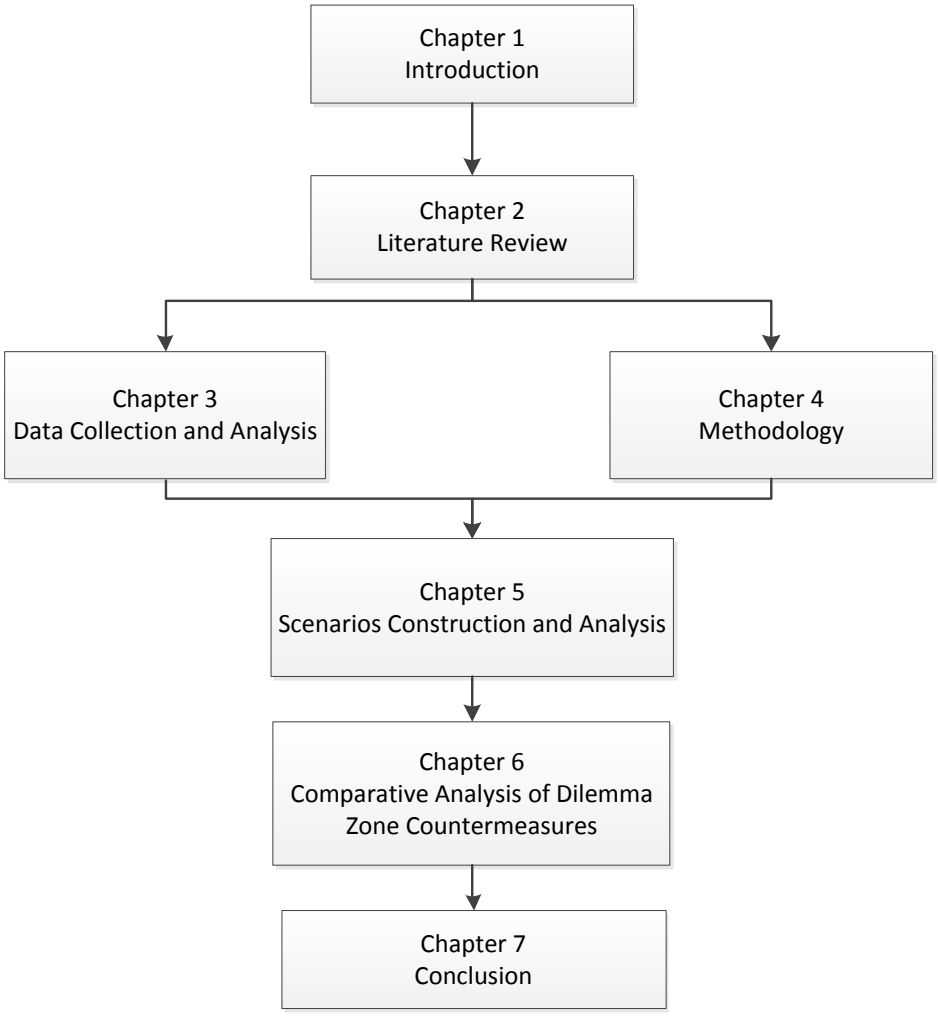


Figure 1.1 Organization of the thesis

CHAPTER 2 LITERATURE REVIEW

Traffic signals are used to direct traffic to stop or to proceed, and separate conflict movement oat intersections. Nowadays, the signal control systems are designed to increase the traffic safety of the overall network. The regular sequence of traffic signal indications is green→yellow→red. The yellow signals indicate the driver need to stop unless he/she cannot stop safely.

2.1 Dilemma Zone

In order to eliminate the conflict between vehicles of different directions, traffic lights were introduced into the road traffic system. By clearing one traffic stream of the intersection before the onset of the green signal of the conflicting stream, the yellow phases are designed to provide an orderly transition. Steady yellow signal indications are displayed after every green phase according to the MUTCD (Manual on uniform traffic control devices: for streets and highways, 2009). The yellow duration is calculated as the following equation (Pline, 1999)：

$$CP = t + \frac{v}{2a \pm 64.4g} + \frac{w + L}{V} \tag{2-1}$$

where:

CP = non-dilemma change period (Change + Clearance Intervals)

t = perception-reaction time (nominally 1 sec)

V = approach speed, m/s [ft/s]

g = percent grade (positive for upgrade, negative for downgrade)

a = deceleration rate, m/s2 (typical 3.1 m/s2) [ft/s^2 (typical 10 ft/s^2)]

W = width of intersection, curb to curb, m [ft]

L = length of vehicle, m (typical 6 m) [ft (typical 20 ft)]

Since the 1960s, traffic engineers have been studying yellow indication related problems. Two types of Dilemma Zone occur due to the yellow phase at signalized intersections (Figure 2.1). The first type of Dilemma Zone may arise because of the insufficient length of yellow as well as all red intervals (Herman and Liu, 1996). In 1960, Gazis et al. (1959) defined the "amber Light Dilemma" as the situation that a driver can neither stop safely nor be able to cross the intersection before the red phase. The yellow interval is usually 3s to 6s duration or longer on approaches with higher speed according to the Manual on Uniform Traffic Control Devices (MUTCD, 2009).

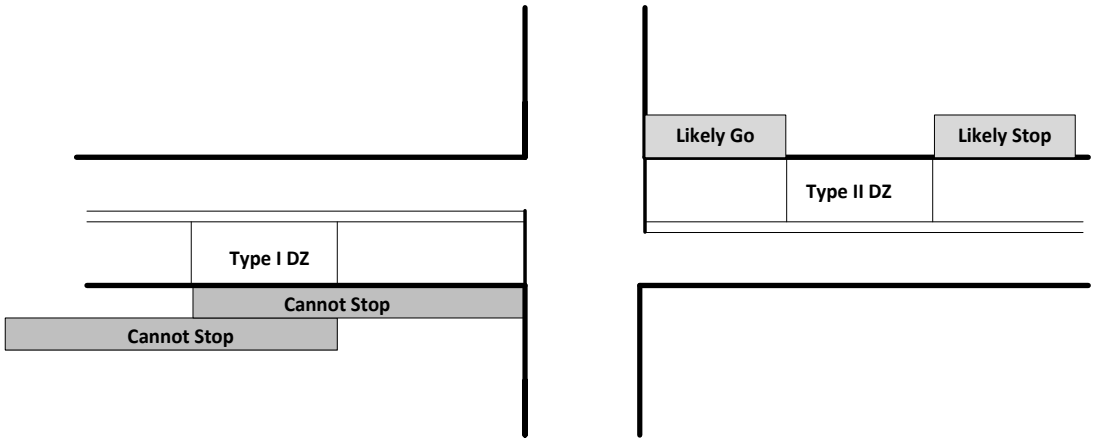


Figure 2.1 Two type of dilemma zone

The second type of Dilemma Zone, which is known as the "indecision zone", is caused by the difference of the driver behavior (Zegeer and Deen, 1978). The problem of Type II Dilemma Zone is more prevalent at high speed intersections because of the greater crashes potential which encouraged studies of the driver behavior at high speed intersections (Hurwitz, 2009). The

increases of Type II Dilemma Zone length leads to a greater risk of rear-end crashes (Zaidel and Klein, 1985; Liu et al., 2007; Newton et al., 1997). There are some attempts to locate the position of Type II Dilemma Zone. In 1978, the Dilemma Zone was defined as in response to the yellow indication the area upstream from the stop line between which 10 percent and 90 percent of the drivers will stop(Zegeer and Deen, 1978). The length of this zone is typically between 2.5s and 5.5s upstream of the stop line (James et al., 2002). In this study, only the Type II Dilemma Zone is considered.

Rear-end crashes and RLR violations are two types of risk situations that are related to the dilemma zone. Some rear-end crashed studies have been conducted based on historical data (Kostyniuk, 1998; Abdel-Aty and Abdelwahab, 2004;Yan et al, 2005). Previous research has revealed that a greater number of rear-end crashes occurs with an increase in the Type II dilemma length (Newton et al., 1997; Kikuchi and R. Riegner, 1992).In 2000, Bryan E. Porter (Porter and England, 2000) observed 5,112 drivers who entered traffic controlled intersections in three cities. The result shows that 35.2% of the observed light cycles had at least one red-light running violation prior to the onset of the opposing traffic. A rear-end crash happens when two successive drivers make conflicting decisions at the onset of the yellow signal (Lum and Wong, 2003).

In terms of driver behavior, some studies have investigated the driver behavior in the dilemma zone. Specifically, classification analysis of drivers' stop/go decisions and RLR violations (Elmitiny et al., 2010) concentrated on predicting the drivers' stop/go behavior when they encountered a signal change from green to yellow and predicted the RLR violations. This project provided evidence in understanding drivers' stop/go decisions as well as providing the field data

to investigate related driver behaviors. The stop/go probability of the drivers was modeled as a function of the distance or gap from the stop line using the logistic regression model technique (Bader and Axhausen, 2004; Newton et al., 1997; Papaioannou, 2007; Yan et al., 2007).

2.2 Advanced Warning Measures

For decades, traffic engineers have come up with some different countermeasures to improve the situation. There are two types of methods that were proposed to help solve the dilemma during the yellow interval: advanced time-warning methods and advanced distance-warning methods. The concept of advance warning is to alert drivers for the potential need to stop, in order to reduce the risk situations.

During the 1960s, some countries introduced some kinds of traffic signal modifications, such as adding flashing green or flashing amber phases, employing green signal countdown display and using advanced warning flashers (Table 2.1). In terms of flashing green or flashing yellow, in some countries, the flashing green/yellow phases are implemented at the end of green phases to give drivers advanced coming yellow indication information. Instead of the regular signal-timing program the overlapping signal would be set as green→/flashing green/flashing yellow→yellow →red. The flashing green signals indicate that the green phases will end, but it is still part of the green phases. In Israel, since the 1960s a flashing green measure has been put into practice, which has 3s (minimum) flashing green signal at 2 Hz before the yellow indication (D. M. Zaidel D. Mahadlel, 1985). The flashing green is regarded as part of the green phase in the signal timing. In the United States, Chicago experimented green-amber and amber phases at some intersections, and in Arizona, a new Traffic Light Change Anticipation System (TLCAS) was carried out (Newton et al., 1997). Previous research found that the green flashing or flashing

yellow method might reduce right angle collision and severity of maximum accelerations and decelerations. Nevertheless, because of the increase of the indecision duration, the flashing green program has high variability in response time and decision, which may cause that rear-end crash risk increases (Factor et al., 2012; Köll et al., 2004; Newton et al., 1997). Both statistical studies and driving simulator studies were implement to analyze the driver behavior for the flashing green or flashing yellow indication (Knodler. Jr and Huiwitz, 2007; Knodler Jr. et al., 2001) .

Meanwhile, some other advanced time-warning methods have been proposed in recent years, The countdown device is widely used in Asia to provide countdown timing and help driver make decisions (Chiou and Chang, 2010; Halim and Lum, 2006; Li and Ni, 2014). As for the Advanced Warning Flashers (AWFs), previous research revealed the AWFs can reduced the crash frequency and severity. However, there is no statically significant was found, and it might increase the operation speed and RLR violations (Carroll et al., 2003). Of all the different types of time-advanced warning methods, nowadays, only a few countries still use these systems (Chiou et al., 2005).

Table 2.1 Different types of dilemma zone countermeasures

Type	Dilemma Zone Countermeasures	Description	Application
Advanced Time-Warning Method	flashing green/yellow phases	Add a flashing green/yellow phase at the end of the green phase	In Israel, the countermeasure has been used for more than 40 years(Factoret al., 2012)
	Green Signal Countdown Display (GSCD) or Red Signal Countdown Display (RSCD)	Provide a green or red countdown timing	Widely installed in Asian countries and areas, such as Malaysia, Singapore, mainland of China, Taiwan, etc.(Li and Ni, 2014)
	Advanced Warning Flasher (AWF)	Provide warning of the signal changing with a flasher	Have been used in the United States and Canada
Advanced Distance-Warning Method	Pavement marking	Add a pavement marking "SIGNAL AHEAD" upstream of the intersection	A few practical use in the United States
Others	Red light camera	Effective in reducing RLR violations, but some additional rear-end crashes might happen	Widely used in many countries as a supplement to police efforts to enforce traffic signal laws (Retting et al, 1999)

With respect to the distance-warning methods, a pavement marking was introduced into one intersection in Florida in order to help the drivers make correct decisions during yellow periods. Yan et al. (2009) studied driver behavior at the intersection where has a pavement marking with

a word message “SINGAL AHEAD” to help drivers decide whether to stop or go at the onset of the yellow indication (Figure 2.2).

Figure 2.2 Pavement marking countermeasure

At the onset of the yellow indication, if the vehicle is located the upstream of the pavement marking, he/she is encouraged to stop during the yellow interval. Otherwise, if the vehicle is located downstream of the pavement marking, he/she can cross the intersection safely. The driving simulator experiment results indicated that the pavement marking countermeasure has positive effect on intersection safety and can help reduce the indecision region. The smaller indecision region will normally imply a reduction of rear-end crashes risk and RLR violations. (Yan et al., 2007).

Other countermeasures are also been widely used, such as the red light camera enforcement, which is applied as a supplement to police efforts to enforce traffic signal laws (Retting et al, 1999).

A major disadvantage of those dilemma zone countermeasures is the assumption of the static dilemma zones. In recent years, transportation researchers realized the location and the length of the dilemma zone are dynamic and may be affected by many factors, such as the different speed

of the approaching vehicles, driver reaction times, vehicle acceleration and deceleration rates, and the yellow phase duration (Liu et al., 2007). Based on this finding, some new countermeasures are proposed, which use the measured speed to determine the individual dilemma zone instead of using the assumed speed (Bonnesonet al., 2002; Tarko et al., 2006). However, most of the practical countermeasures have not considered this disadvantage.

2.3 Intersection Simulation based on Cellular Automata Model

Using traditional traffic models to describe the microscopic behavior of vehicles can be a very time consuming and complex process. Recently, with the rapid development of the computation technologies, many Cellular Automata (CA) based models were developed.

In 1980s, Wolfram (Wolfram, 1983) developed the first well recognized CA model and introduced the well-known "184 model". Negal and Schreckenberg (1992) developed a CA models for traffic flow simulation, which are easier to implement on computers for numerical investigations. In this model, a lane is consisted by a number of one-dimensional cells, and is updated according to pre-defined transition rules, which includes acceleration step, deceleration step, randomization step and updating step. When employing the CA model into traffic flow simulation, a lane is represented by a series of one-dimensional cells, and the cell are all equal size. The underlying structure is composed by a discrete lattice of cells with one type of topology, such as rectangular and hexagonal (Maerivoet and Moor, 2005). Each cell may either empty or be occupied by one vehicle. The cell's neighborhood determines the evolution of the cell. Position and velocity of the n^{th}vehicle are represented by x_n and v_n. Also, the headway between the vehicle and the vehicle in front is d_n, which equals to $x_{n+1} - x_n$. The velocity of

each vehicle is between 0 and v_{max}. Successful application has been employed to simulate the urban traffic flow (Clarridgea and Salomaab, 2010; Jiang and Wu, 2006; Han and Ko, 2012).

In order to evaluate the crash risks, Boccara et al. (1997) represented the Risky Situations (RS) using a CA model. Jiang et al.'s studies (Jiang et al., 2003; Jiang et al., 2004) demonstrated that RS reflected well the occurrence of rear-end crashes. Two types of rear-end crashes are analyzed in this study: risks caused by a stopped car, risks caused by a non-stopped car. High deceleration was also described in this situation.

However, of all the previous findings, relatively few studies use the drivers' stop/go decision database and compare different countermeasures to explore the propensity of accidents and violations. There is a critical need for research to quantify driver stop/go behavior and analyze the advantage as well as disadvantages of different countermeasures. Ding et al. (2014) employed a CA model into the intersection's RS analysis where decision tree model was used to predict the drivers' stop/go decisions based on field inventory data. The CA model's main parameters' calibration was based on the results of the decision tree modeling. Since the dependent variable (stop/go decision) is dichotomy, a logistic regression model is proper to analyze the relation between the dependent variable and the factors, to rank the relative importance of the independent factors, and to predict the stop/go decisions. The research applies logistic regression model to predict drivers' stop/go decisions, and compares three types of scenarios (the typical intersection signal, the intersection with the flashing green indication as well as the intersection with pavement marking before the stop line) Also, a new countermeasure (PMAIC) that combined both time and distance solution is proposed and its effectiveness is evaluated to further improve the intersection's safety during the yellow interval.

CHAPTER 3 DATA COLLECTION AND ANALYSIS

3.1 Observation Site Description and Data Collection

The data collection conducted in Orlando, Florida. The studied intersection is located at the Northwest corner of the University of Central Florida, which plays a major role in stimulating economic and residential development in Orlando. The studied intersection is a four-legged intersection between Corporate Blvd. & Gemini Blvd. running from east-west and Alafaya Trail running north-south (Figure 3.1). The intersection is a signalized intersection. The yellow interval is 4.3 seconds and the all-red interval is 1 second.

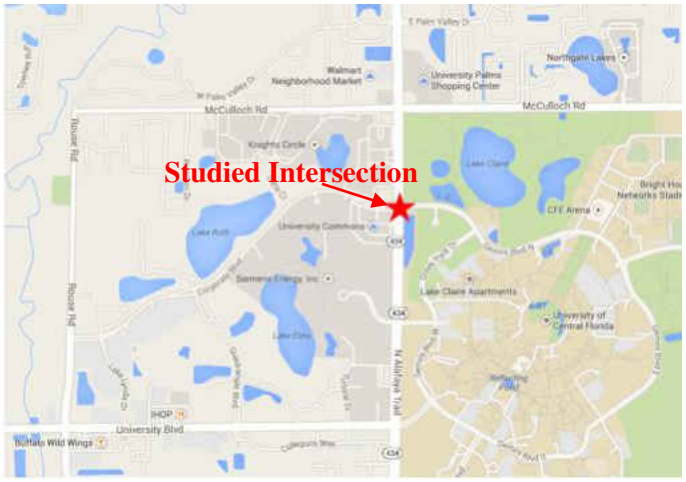


Figure 3.1 Site location map ("google map", 2014)

The northbound approach is specifically considered for the study (Figure 3.2).

(a)

(b)

Figure 3.2 South approach of the studied intersection

(a) View from the intersection (b) View into the intersection

When off the intersection, the southbound of Alafaya Trail are five-lane divided traffic: three for direct movements, one exclusive left turn lane and one exclusive right turn lane. The northbound is three-lane divided traffic. The eastbound on Corporate Blvd. and the northbound on Gemini Blvd. are two-lane divided traffic. The existing posted speed limit on Alafaya Trail is 45 miles per hour (mph) (Figure 3.3).

Thirty-six hours of video, which includes 28 off-peak hours (1:30pm-4:30pm) and 8 peak hours (4:30pm-6:00pm), were filmed during the weekdays. Using Adobe Premiere Pro software to extract data from videos, 1292 vehicles' behavior was recorded, which does not include the

vehicles forced to stop by the vehicle in front. Due to the small sample size of light truck vehicles, data of this vehicle type were excluded from the database in this study.

Eight variables were obtained from the video:

- DISTANCE (in ft): car's distance from the intersection at the onset of the yellow indication;
- SPEED (in mph): car's operating speed at the onset of the yellow indication;
- ST_GO: driver's stop/go decision (stop = 0; go = 1);
- Y_TIME (in seconds): time elapsed from the onset of the yellow until the car entered the intersection, if the car crossed the intersection;
- RLR: whether the going car ran a red light or not (no = 0; yes = 1);
- LD_FL: whether the car was in a leading position or a following position in the traffic flow (leading = 0; follow= 1); if headway was shorter than 1 s the car was considered following in the platoon;
- L_POSITION: the car's lane position (left lane = 0; middle lane = 1; right lane = 2);
- V_TYPE: vehicle type [passenger car (PC) =0; light truck vehicle (LTV)=1; ;lager size vehicle (LSV)=2].

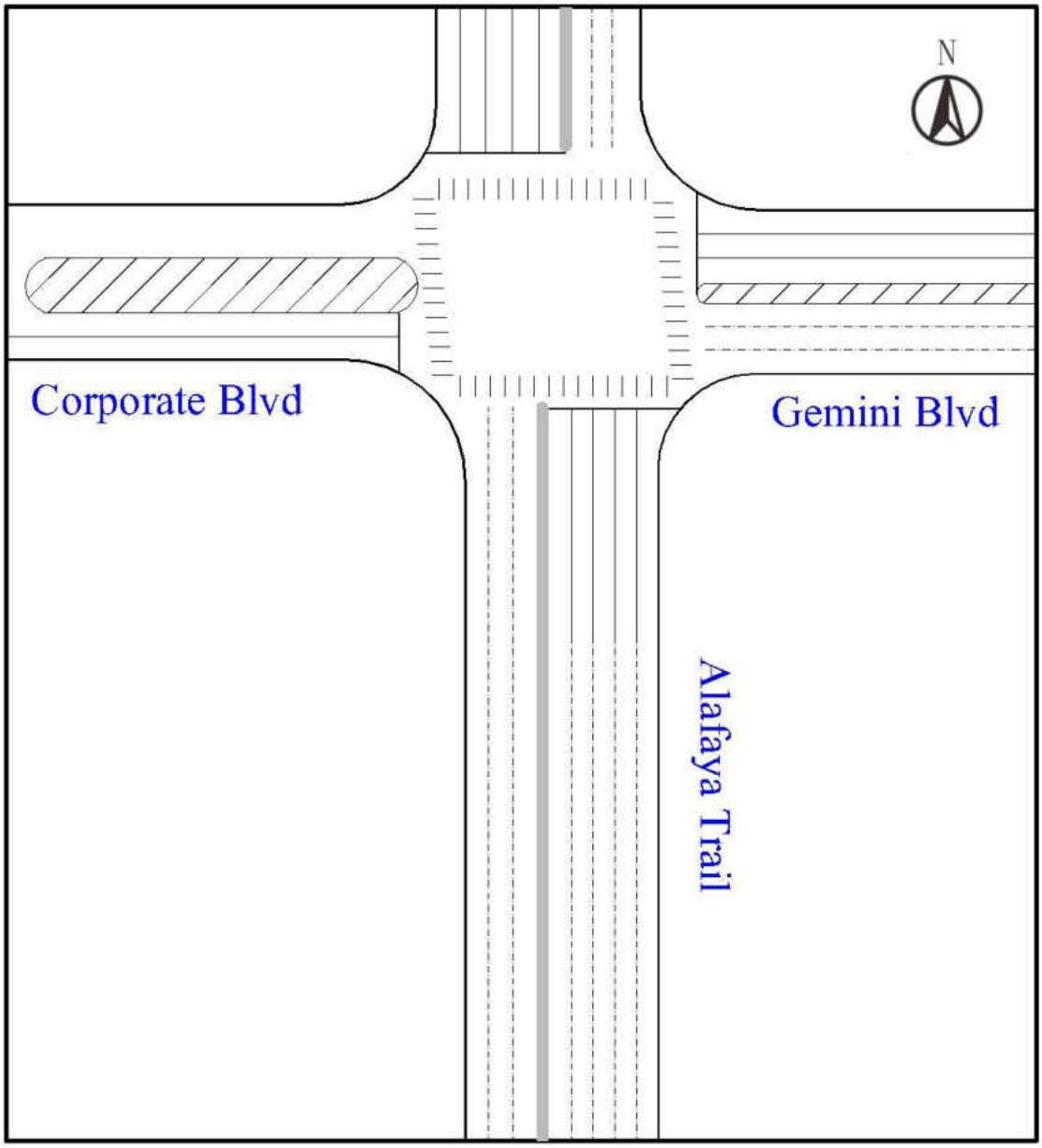


Figure 3.3 Condition diagram

3.2 Observation Results and Data Analyses

3.2.1 Drivers' Stop/Go Decisions

The ST_GO variable describes the drivers' decisions at the onset of the yellow indication. "Stop" means the driver chose to stop during the yellow interval, while "go" means the driver decided to cross the intersection at the yellow interval. Five hundred and eight-five go decisions as well as six hundred and seventy-nine stop decisions were observed. Both of the decisions account for about 50% of all observations.

A logistic model was employed to analyze the importance of different independent variables for the drivers' stop/go decisions (Table 3.1). Five variables were considered during the variable selection, which include SPEED, DISTANCE, LD_FL, L_POSITION and V_TYPE.

Table 3.1 Independent variables for the stop/go decision

Parameter	Estimate	Standard Error	Wald χ^2	Pr > ChiSq
SPEED	0.2118	0.0183	134.302	<.0001
DISTANCE	-0.0246	0.00145	286.449	<.0001
LD_FL	1.0327	0.1622	40.5204	<.0001
L_POSITION	0.0201	0.0991	0.0413	0.8389
V_TYPE	0.2209	0.1577	1.9624	0.1613

According to the results of the logistic regression analysis, SPEED, DISTANCE and LD_FL variables have significant impact on drivers' stop/go decisions at 0.05 significance level; however, the other 2 variables (L_POSITION & V_TYPE) do not have significant influence on

drivers' stop/go decisions. Thus, three significant variables (SPEED, DISTANCE and LD_FL) were chosen to be the main factors for predicting the drivers' decisions. The results are consistent with the results of the previous study (Elmitiny et al., 2010).

3.2.1.1 Main Factors of Drivers' Stop/Go Decisions

1) Speed Variable

The speed limit of the north approach is 45 mph. The mean speed is 48.2 mph, which is slightly higher than the speed limit and lower than the lead vehicles' mean speed 49.0mph. The range of the operating speed is 25mph to 63mph, and the standard deviation of the speed is 5.0mph. Most of the operating speeds of the vehicles are at the 45mph to 55mph interval, which accounts for 73.2% observations (Figure 3.4). The leading vehicle speeds follow Normal Distribution N~ $(49.5,4.9^2)$, which is considered to be the expected speed distribution for leading vehicles in this simulation research.

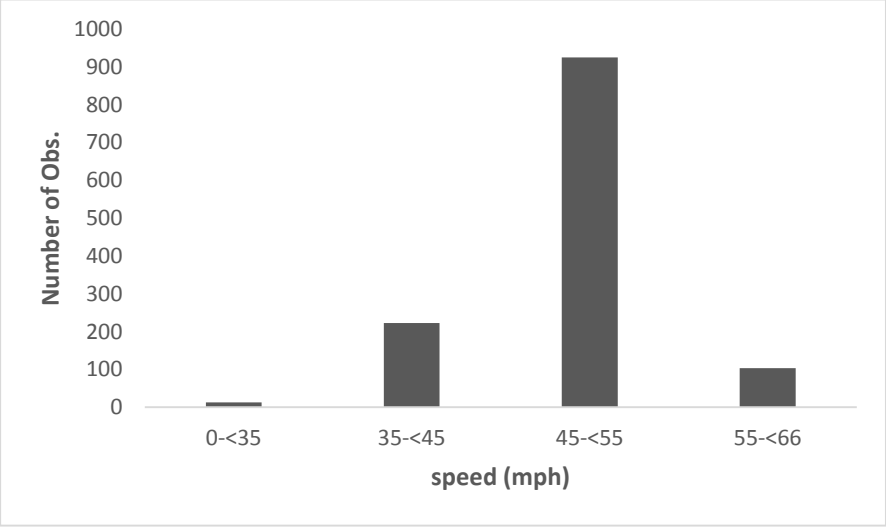


Figure 3.4 Number of observations in different speed intervals

The mean operating speed of vehicles that made go decisions (Mean=49.93, SD=4.99) is statistically higher (p-value=0.000) than the mean operating speed of vehicles that made stop decisions (Mean=47.79 SD=4.83).

In this study, the speed variable is divided into 3 groups.

Group 1: (0 mph, 45mph),

Group 2: [45 mph, 53 mph),

Group 3: [53 mph, 66mph).

Table 3.2 lists the descriptive statistics of drivers' stop/go decisions at the onset of the yellow signal by speed factor. Statistical results show that people who drive in different speed groups make the stop/go decisions differently ($\chi_{3,1264} = 407{,}173$, p=0.000). With the increase of the speed, the probability that driver choose to stop will also increase.

Table 3.2 Descriptive statistics of stop/go decision by speed factor

Level	Statistics	Stop/go Decision		
		Stop	Go	Total
0-<45	Count	158	78	236
	% Within SPEED	66.95	33.05	100.00
	% Within stop/go	23.27	13.33	18.67
45-<53	Count	456	369	825
	% Within SPEED	55.27	44.73	100.00
	% Within stop/go	67.16	63.08	65.27
53-<66	Count	65	138	203
	% Within SPEED	32.02	67.98	100.00
	% Within stop/go	9.57	23.59	16.06
Total	Count	679	585	1264
	% Within SPEED	53.72	46.28	100.00
	% Within stop/go	100.00	100.00	100.00

In Group 1, 33.05% drivers only choose to go. The percentage increases to 44.72 when the driver travels at 45 mph to 53 mph. When the vehicle's speed is 53 mph to 66 mph, the probability to go is 67.49%, which is significantly higher than the probabilities of group 1 and group 2. Regardless of the distance factor and other factors, about 50% of the drivers who drive at the speed of 45 mph to 53mph will choose to stop.

2) Distance Variable

The mean distance of the vehicles to the stop line is 319.3ft at the beginning of the yellow phases and the standard deviation is 80.2ft. The minimum distance is 160.0ft and the maximum speed is 480.0ft. As we can see in Figure 3.5, most of the observations were in 200ft to 400ft distance region.

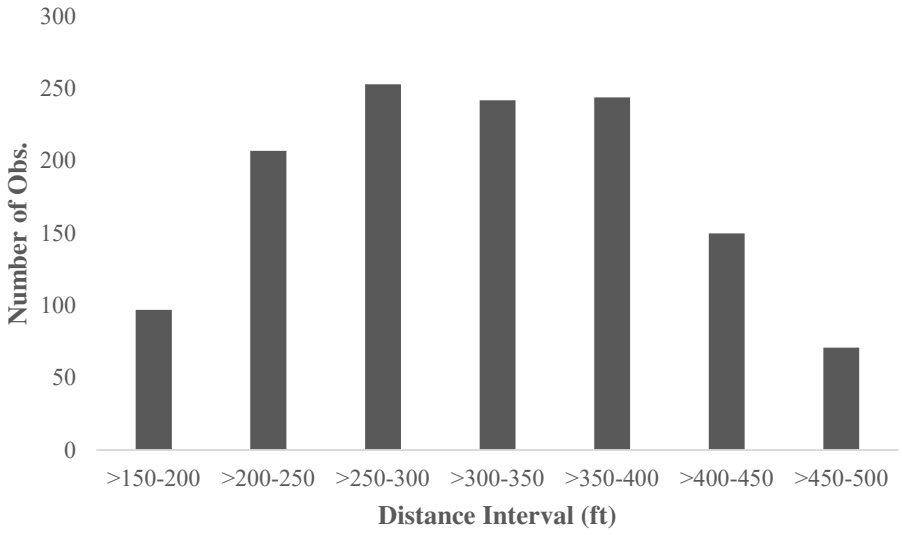


Figure3.5 Number of observations in different distance intervals

The distance to the stop line has a negative effect on the percentage of drivers who decide to cross the intersection. About 90% drivers chose to go if they are within 220ft to the stop-line. Also, when the distance is more than 400ft, the probability will drop to below 10% (Figure 3.6).

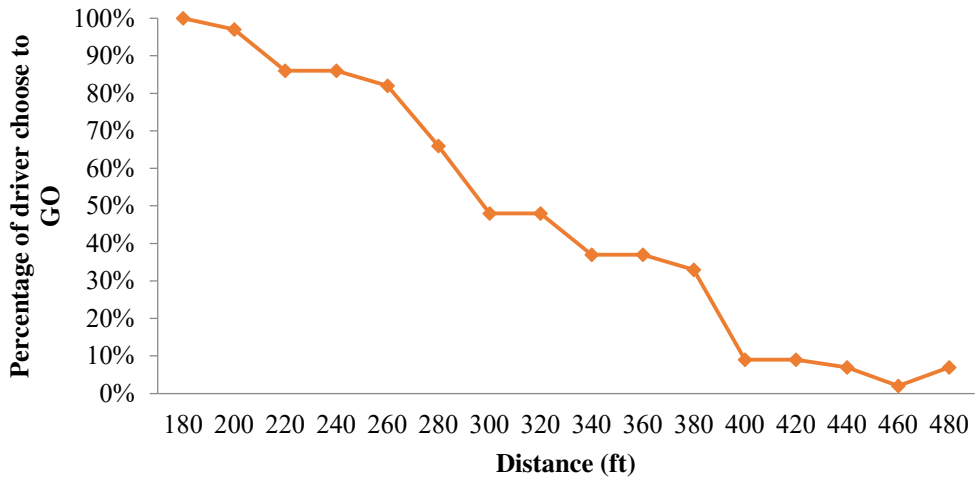


Figure 3.6 Stop/go decisions with different distance to the stop-line

The vehicles' distance to the stop line is divided into 4 groups (ft.):

Group 1: (0,280);

Group 2: [280,390);

Group 3: [390,430);

Group 4: [430,480)

Table 3.3 lists the descriptive statistics of drivers' stop/go decisions at the onset of the yellow signal by distance factor. The statistical test demonstrates a significant difference from the different speed groups ($\chi_{2,1264} = 55.863$, p=0.000). In Group 1, most of the drivers will choose to cross the intersection instead of stop. The number drop significantly in group 2, which only about 40% drivers will decide to go. The go percentage of drivers who is more than 390ft away from the stop line is below 10%. The trend is logical that the driver is more likely to cross if he/she is closer to the stop line at the onset of the yellow signal.

Table 3.3 Descriptive statistics of stop/go decision by distance factor

Level	Statistics	Stop/go Decision		
		Stop	Go	Total
0-<280	Count	75	349	424
	% Within DISTANCE	17.69	82.31	100.00
	% Within stop/go	11.05	59.66	33.54
280-<390	Count	348	217	565
	% Within DISTANCE	61.59	38.41	100.00
	% Within stop/go	51.25	37.09	44.70
390-<430	Count	120	11	131
	% Within DISTANCE	91.60	8.40	100.00
	% Within stop/go	17.67	1.88	10.36
430-<480	Count	136	8	144
	% Within DISTANCE	94.44	5.56	100.00
	% Within stop/go	20.03	1.37	11.39
Total	Count	679	585	1264
	% Within DISTANCE	53.72	46.28	100.00
	% Within stop/go	100.00	100.00	100.00

3) LD_FL Variable

Driver behavior of the leading and following vehicle is different. Hurwitz (2009) analyzed the driver responses and pointed out the difference between lead and follow vehicles. During the data collection, 565 leading vehicles and 699 following vehicles were recorded.

Table 3.4 lists the descriptive statistics of drivers' stop/go decisions at the onset of the yellow signal by LD_FL factor. The position in platoons also has a significant difference on driver behavior ($\chi_{1,1264} = 93.104$, p=0.000). The table 3.4 also indicates that the following vehicles are more prone to cross the intersection compared with the leading vehicles.

Table 3.4 Descriptive statistics of stop/go decision by LD_FL factor

Level	Statistics	Stop/go Decision		
		Stop	Go	Total
Lead	Count	388	176	564
	% Within LD_FL	68.79	31.21	100.00
	% Within stop/go	57.14	30.09	44.62
Follow	Count	291	409	700
	% Within LD_FL	41.57	58.43	100.00
	% Within stop/go	42.86	69.91	55.38
Total	Count	679	585	1264
	% Within LD_FL	53.72	46.28	100.00
	% Within stop/go	100.00	100.00	100.00

The speeds of vehicles at different positions are significantly different. The mean speed of leading vehicles (Mean=49.52, SD=4.93) is statistically higher than the following vehicles (Mean=48.19 SD=5.01) with p-value equal to 0.000.

3.2.1.2 Other Factors of Drivers' Stop/Go Decisions

1） Y_TIME Variable

If the vehicle chose to enter the intersection at the onset of the yellow indication, the time elapsed from the onset of the yellow until the car entered the intersection was recorded. The mean time is 3.9 seconds and the standard deviation is 0.8 second. The minimum time is 2.1 seconds, while the maximum time is 7.2 seconds. Detailed analysis of Y_TIME will be conducted at chapter 3.2.2 (Red-Light Running Violation).

2) V_TYPE Variable

There were 538 light trucks and 726 passenger cars were observation during the data collection (Figure 3.7). The statistical analysis does not show significant difference between passenger car and light truck vehicles ($\chi_{1,1264} = 1.576$, p=0.209).

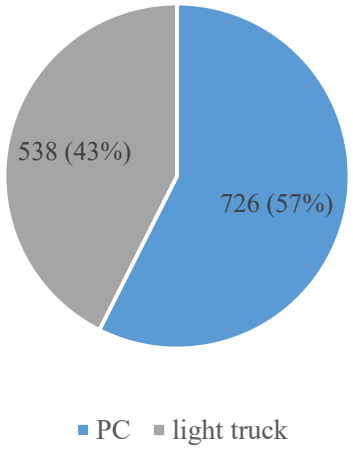


Figure 3.7 Number of observations by different types of vehicles

The mean speed of passenger car is 49.0mph and the mean speed of light truck is 48.4mph (Figure 3.8). Significant difference has been found by statistical analysis at 0.05 significance level (p-value=0.031).

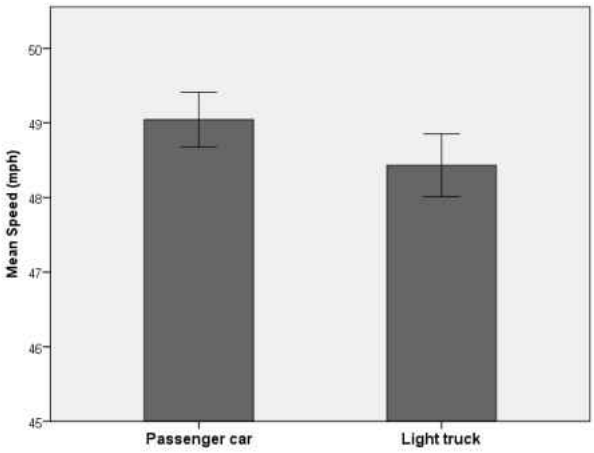


Figure 3.8 Mean of the speed by different vehicle types

3) L_Position Variable

There are three lanes at the studied approach. The middle lane has the highest mean speed, while the left lane has the lowest mean speed. However, the right lane has the lowest standard deviation of the operating speed (Table 3.5).

Table 3.5 Speed by different lane position

	Left Lane	**Middle Lane**	**Right Lane**
Mean (mph)	47.59	49.77	49.07
Std. Deviation (mph)	5.28	4.80	4.63

Table 3.6 shows the contingency table of different lane positions. There is no significant difference of the drivers' stop/go decisions between vehicles at different lanes ($\chi_{2,1264} = 1.287$, p=0.525).

Table 3.6 Contingency table of stop/go decisions by different lane positions

	Go	**Stop**
Left Lane	202	255
Middle Lane	215	235
Right Lane	168	189

3.2.2 Red-Light Running Violation

Typically, there are two types of Red-light running violations (RLR) (Federal Highway Administration). The first type of RLR, which is referred as "permissive yellow" rule, means the driver can enter the intersection legally during the yellow interval. The second type is called

"Restrictive yellow" rule, which forbid the driver enter or in the intersection on red interval. The first type of rule is more commonly used in the United States.

Two hundred and seventeen red-light running violations were observed during the data collection. Table 3.7 shows the descriptive statistics of RLR violations. The speed group 2, distance Group 2, following vehicles, vehicles in the right lane has relatively higher percentage of RLR violations. Most of the vehicles that have the RLR violation need about 4 seconds to 5 seconds to cross the intersection at the onset of the yellow indication, which indicates driver are more prone to make false stop/go decisions when they has 4s to 5s elapsed time to enter the intersection.

Table 3.7 Descriptive statistics of RLR violation by different factors

Factor		Statistics	RLR
speed group	0-<45	Count	35
		Percentage (%)	16.1
	45-<53	Count	129
		Percentage (%)	58.9
	53-<66	Count	53
		Percentage (%)	24.4
Distance group	0-<280	Count	39
		Percentage (%)	18.0
	280-<390	Count	160
		Percentage (%)	73.7
	390-<430	Count	10
		Percentage (%)	4.6
	430-<480	Count	8
		Percentage (%)	3.7
Lead/Follow	lead	Count	52
		Percentage (%)	24.0
	follow	Count	165
		Percentage (%)	76.0
Lane position	Left Lane	Count	71
		Percentage (%)	32.7
	Middle Lane	Count	88
		Percentage (%)	40.6
	Right Lane	Count	58
		Percentage (%)	26.7
Vehicle type	Passenger Car	Count	124
		Percentage (%)	57.1
	Light Truck	Count	93
		Percentage (%)	42.9
Elapse time	>4-5	Count	163
		Percentage (%)	75.1
	>5-6	Count	52
		Percentage (%)	24.0
	>6-7	Count	2
		Percentage (%)	0.9

Table 3.8 demonstrates the logistic regression analysis for RLR violations. Five factors are considered. Three factors (follow or lead, speed and distance) show a significant impact on the presence of RLR violations. Distance variable has the highest impact on the presence of RLR

violations, while the lead or follow position has the least impact on the presence of the RLR violations. However, the vehicles' lane position and the vehicle type do not show significant relationship with the present of the red-light running violation. The parameter estimates indicates the follow vehicles are more prone to have RLR violations. Also, the distance has a negative effect on reducing RLR violations, while the speed has a positive effect on reducing RLR violations. Compared the RLR violations of the leading vehicles and following vehicles, significant statistically difference has been observed ($\chi_{1,1264} = 45.239$, p=0.000), which indicates the following vehicles are more prone to have a RLR violations than the leading vehicles.

Table 3.8 Parameter estimates of the logistic model for RLR violation

Parameter	Estimate	Standard Error	Wald χ^2	Pr > ChiSq
SPEED	0.0578	0.0164	12.4733	0.0004
DISTANCE	0.00331	0.00103	10.2341	0.0014
LD_FL	1.3737	0.1836	55.9846	<.0001
L_POSITION	0.0486	0.0980	0.2453	0.6204
V_TYPE	0.0527	0.1558	0.1143	0.7353

The distance of 340ft to 370ft has the most RLR violations. Nearly 60 red-light running violations were observed in that distance region. The number of RLR violations shows an increasing trend in the distance interval of 220ft to 370ft, and shows a decreasing trend when the distance to the stop line is larger than 370ft away from the stop line (Figure 3.9). However, the lane position does not have significant difference on RLR violations ($\chi_{2,1264} = 2.873$, p=0.238).

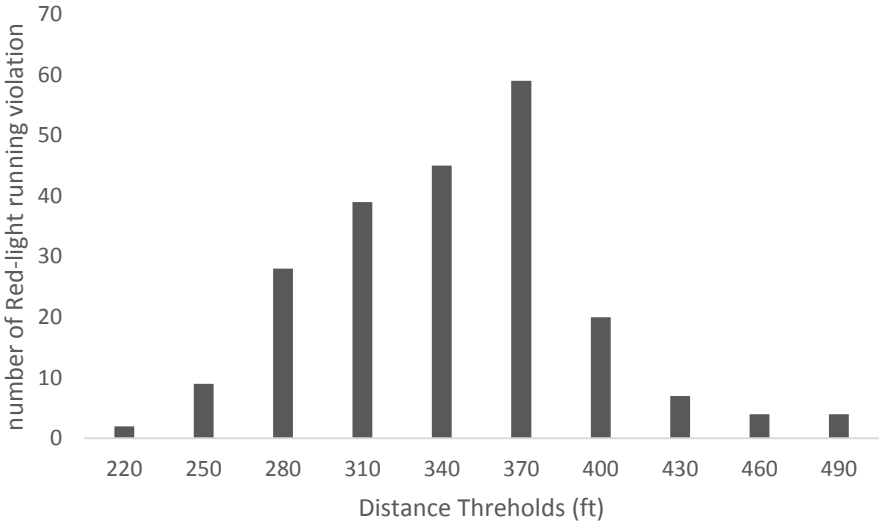


Figure 3.9 Distribution of RLR violation by distance interval

The RLR vehicles (Mean=49.67, SD=5.42) have a higher (p-value=0.008) mean speed than the vehicles without RLR violations (Mean=48.60, SD=4.91). The results indicate that the drivers who have an RLR violation can be more aggressive than the other drivers (Figure 3.10).

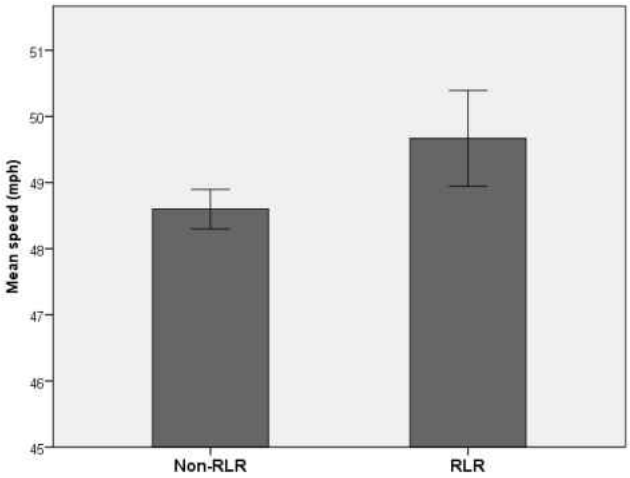


Figure 3.10 Mean of the speed by the presence of RLR violation

Compared the presence of the RLR violations by different position in a traffic platoon, there is significant difference between the lead vehicles and the following vehicles ($\chi_{1,1264} = 32.532$, p=0.000).

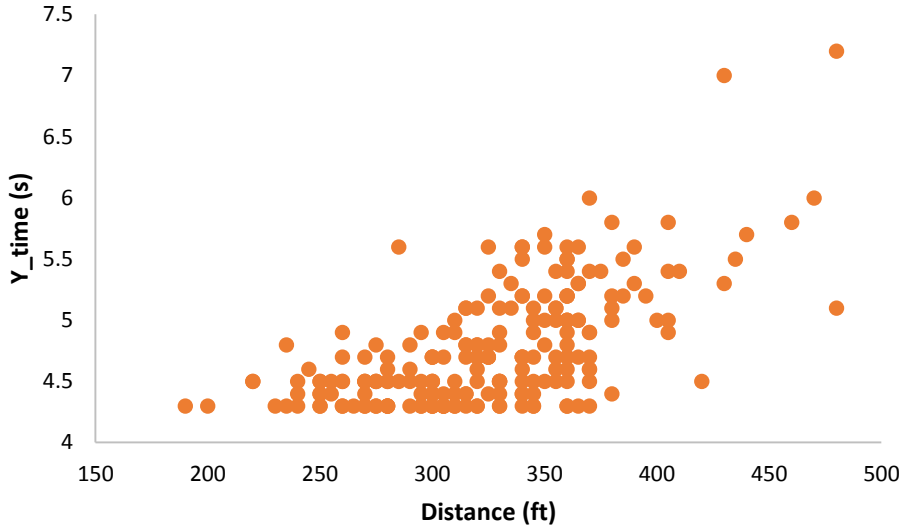


Figure 3.11 Red-light running violations

Figure 3.11 illustrates the elapse time and the distance to the stop line of the vehicles, which had the RLR violations. From Figure 3.11 we can see, most of the violation happen at the 4s to 6s elapse time to enter the intersection and 250ft to 370ft away to the stop line. The results consist with previous studies (Bonneson et al., 2002).

CHAPTER 4 METHODOLOGY

4.1 Stop/go Decision Rule

In statistics, a logistic regression is a type of statistical classification model and a model to predict a binary or categorical dependent variable based on one or more independent variables. A logistic regression analysis can be employed to describe the relationship between explanatory variables and a response variable. Previous studies have appropriately applied logistic regression analysis to test the significance of observable factors and drivers' characteristics and grouped drivers into different categories (Papaioannou, 2007).

A binary logistic regression is proper to use to explain drivers' stop/go decisions as a function of several factors. According to the analysis in Chapter 3.2, a logistic model can also be used to predict driver behavior. Three factors, which include speed group, distance group and lead/follow position, are used as variables in the logistic regression analysis to predict the drivers' stop/go decisions.

The probability that a driver will decide to cross the intersection is modeled as logistic distribution in (4-1) where g(x)=0 stands for stopping and g(x)=1 stands for crossing:

$$\pi(x) = \frac{e^{g(x)}}{1 + e^{g(x)}} \tag{4-1}$$

The Logit of the logistic regression model is given by Eq. (2):

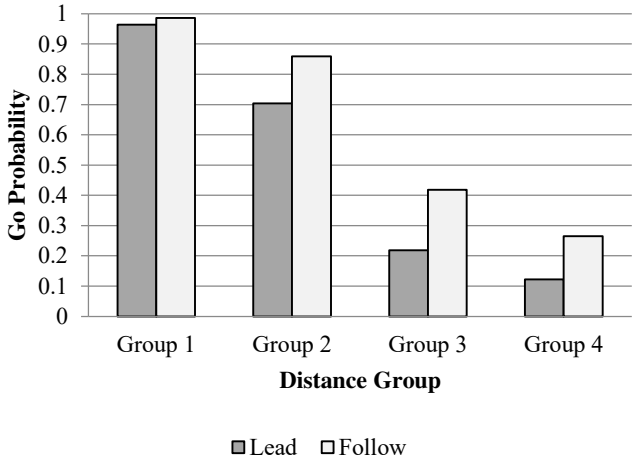
$$g(x) = \ln \frac{\pi(x)}{1-\pi(x)} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \ldots\ldots + \beta_n x_n \tag{4-2}$$

Statistical analysis was performed using SAS and the hypothesis testing was based on a 0.05 significance level (Table 4.1). The logistic model is found to be appropriate for the data (Hosmer-Lemeshow goodness of fit Chi-Square =2.7349, d.f.=8, p-value=0.9499). The ROC area of 0.874 indicated that 87.4% of (go, stop) pairs of decisions were classified correctly by the model, which means that the predictive accuracy is good. The odds ratio of the lead/follow vehicles mean the odds of go decision for follow vehicles is 2.547 times the odds of the go decision for lead vehicles. Meanwhile, according to the Table 4.1, ratio of the odds for distance group 2 and distance group 3 relative to distance group 1 is 4.479 and 26.629 separately. Also, the odds of go decision for speed group 2, speed group 3 as well as speed group 4 are 0.090, 0.011 as well as 0.005 times the odds of go decision for the speed group 1.

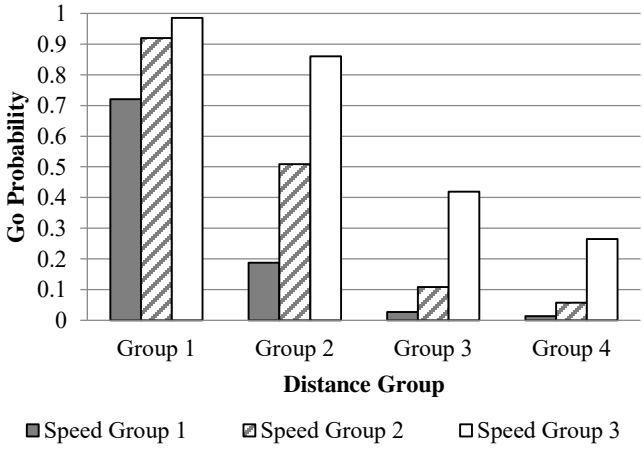
Table 4.1 Model estimation and odds ratios

Parameter	**Estimate**	**Odds Ratio**	**95% Wald Confidence Limits**		**Wald χ^2**	**Pr > ChiSq**
Follow vs. Lead	0.9458	2.547	1.870	3.469	35.8336	<.0001
Speed group						
2 vs. 1	1.4994	4.479	2.974	6.746	51.4817	<.0001
3 vs. 1	3.2820	26.629	14.837	47.793	120.9566	<.0001
Distance group						
2 vs. 1	-2.4108	0.090	0.063	0.128	174.7836	<.0001
3 vs. 1	-4.5557	0.011	0.005	0.022	141.3785	<.0001
4 vs. 1	-5.2498	0.005	0.002	0.013	122.3220	<.0001

Assuming that the speed of a vehicle is 45mph to 53 mph (Group 2), as we can see from Table 4.1, the probability of the lead vehicle driver choosing to go is always lower than the following vehicle and it also drops quicker than the follow group with the increase of the distance. When the lead position car is more than 430ft from the stop line, the probability of the driver choosing to go is only about 10%.



(a)



(b)

Figure 4.1 Drivers' stop/go decisions

Figure 4.1 shows that the go probability for the following vehicle in different speed and distance groups. The driver prefers to go when he/she travels at a higher speed. Even if the car is only

390ft away from the stop line, the relative probability for vehicles below 45mph is only nearly 20%. Vehicles in the following position in the platoon are more prone to go compared with the leading vehicles. Meanwhile, drivers who are in speed Group 1 will be prone to choose to stop if he or she is more than 280ft away from the stop line. If the vehicle is in distance group 4, the drivers would be prone to stop no matter how fast the vehicle is traveling (Figure 1(b))**.**

4.2 Cellular Automata Model

Previously, microscopic simulation of driver behavior is very complex and time consuming. As the rapid development of the computer technology, a number of simulation systems, which includes different types of CA models, have been developed. According to characteristics of the CA model, it is widely used for the traffic flow simulation once it was introduce to the traffic field.

1) Simulation Environment

During the simulation, the lane is made up of cells, which could be empty or occupied by one car. Each cell corresponds to 1.5m, each car occupies 5cells (standard value in CA models). In this study, the simulation environment is set up as an open boundary one-dimensional lattice.

2) Model parameters and variables

A series of parameters are defined in the CA model. Some variables are from the literature, and some variables are calibrated by the field data.

The length of the road is set as 5000 cells, which mean 7500m. t is the number of time step, and 1 time step represents 1 second during the simulation. The simulation covers 1500 seconds (time

steps). The maximum acceleration is 1.5m/s^2 (1 cell/s^2), and the maximum deceleration is 3m/s^2 (2 cell/s^2). The initial operating speed of the vehicles follows normal distribution, which is calibrated by the field data. The expected mean speed of the leading vehicles is set as the input.

Table 4.2 CA model parameters

Notation	Description	Value
L	Road length	5000 (cells)
t	Number of time step	1500 (s)
acc	Maximum acceleration	1 (cell/s^2)
dec	Maximum deceleration	2 (cell/s^2)
v_{int}	Initial speed of vehicle	Normal~ (49.5,4.9^2),

3) Driver Behavior

- General Rules

The response time is referred to the time interval of signal changing and the brake (or acceleration) response. Wortman (1983) found that the average reaction time is between 1.09s and 1.55s, which is consistent with the results of other studies (Chang et al., 1985; Newton et al., 1997; Gates et al., 2006). The Institute of Transportation Engineer's (ITE, 1989) recommended 1.0 second as the brake-response time for yellow interval. In the simulation, the default value for the yellow interval reaction time is 1.0 second.

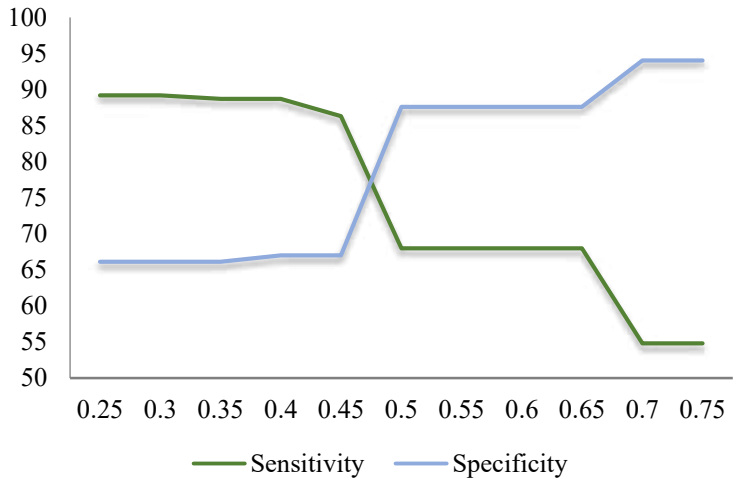


Figure 4.2 Relationship of sensitivity and specificity in the logistic model

Drivers' stop/go decisions are based on the probabilities calculated by the logistic regression. The sensitivity and specificity reach the same value when the probability is equal to 0.48 (Figure 4.2). Therefore, if the probability value is larger than 0.48, the driver will choose to go. Otherwise, the driver will choose to stop. Other driver behavior rules follow the CA model rules (Y. Ding et al., 2014; Jia et al., 2007).

- Randomization

For the drivers who neither have an obvious speed up behavior nor slow down. We assume the slowing down of the speed is caused by the randomization. For all the non-stopping cars' acceleration reveals a normal distribution. Ding et al. (2014) has calibrated the randomization probability at the intersection is $p = 0.16$.

4) Updating Rules

The position and velocity of the vehicles are updated according to the following transition rules:

Step1: Acceleration: If $v_n < v_{max}$, the speed is advance by one, unless the distance to the next vehicle ahead is smaller than $v_n + 1$.

$$v_n \rightarrow \min(v_n+1, v_{max}) \tag{4-3}$$

Step 2: Deceleration: If the n^{th}vehicle's speed will exceed the front vehicle at the next time step (Δt), the velocity of the n^{th} vehicle is reduced by 1.

$$v_n \rightarrow \min(v_n, d_n/\Delta t - 1) \tag{4-4}$$

Step 3: The velocity of each vehicle (if $v_n > 0$) is decreased by one with probability p.

$$v_n \rightarrow (v_n - 1,0) \tag{4-5}$$

Step 4: Update vehicle movement

$$x_n \rightarrow x_n + v_n \times \Delta t \tag{4-6}$$

4) Signal Timing

A fixed time signal control program is set at the intersection. To simplify the simulation process, the intersection signal is set as a fixed timing program with a relatively short circle of 60s including a 25s green phase, a 4s yellow phase and a 31s red phase. In the fixed signal time

program, when signal change from green to amber drivers will make stop/go decision and behave differently under different countermeasures.

4) Risky Situation

Two types of the dangerous situations are considered during the simulation, which includes the rear-end crashes and the red-light running violations. There is no rear-end crash happens during the simulation based on the CA model, thus a concept of risky situation was proposed to describe the rear-end crashes caused by the false behavior of the drivers. Sometimes, the drivers' false behaviors are caused by the false expectations of other drivers. In this study, risky situations can be divided into two types, one is risky situations caused by a stopped car, and the other one is caused by non-stopped cars. Also, a criterion of slam on the brake is defined to describe the situation when the former car encounter emergency (such as signal turning from green to yellow), a risky situation may present because of the inefficient response time. Meanwhile, the presence of RLR violations is also caused by the false decision of the drivers. Thus, another criterion, which described the percentage of drivers' false go decisions, is proposed to compare the potential RLR risk.

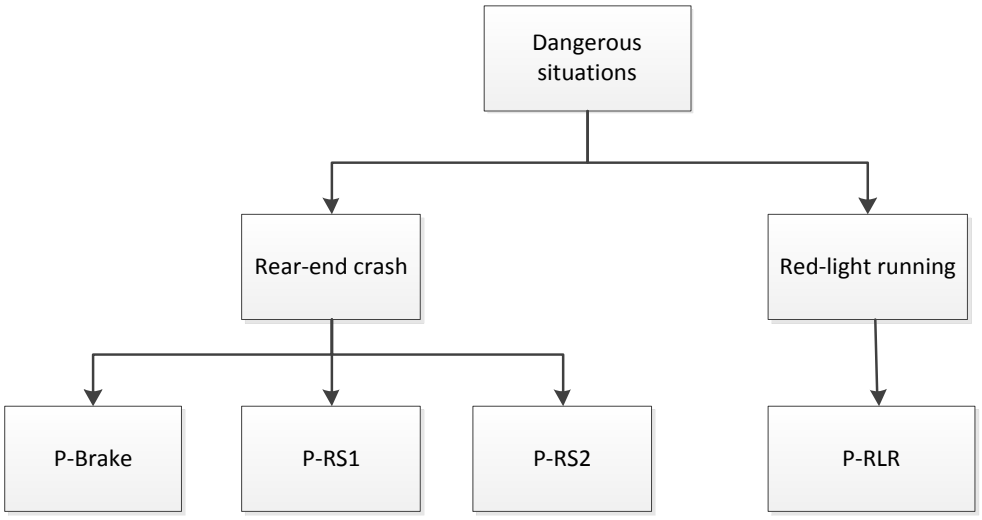


Figure 4.3 Dangerous situations

Four risky situations are analyzed in this study, which includes slam on the brake (situations caused by stopped car, non-stopped cars and Red-Light Running Rate (RLR).

Slam on the brake (BRAKE)

$$(a1)v_n^t - v_n^{t+1} > 2 \tag{4-7}$$

RS caused by stopped cars (RS1)

$$(a1)v_n^t - v_n^{t+1} > 2 and\ d_n^{t+1} = 0, (b1)v_{n+1}^t > 0, (c1)\ v_{n+1}^{t+1} = 0 \tag{4-8}$$

RS caused by non-stopped cars (RS2)

$$(a1)v_n^t - v_n^{t+1} > 2 and\ d_n^{t+1} = 0, (b1)v_{n+1}^t > 0, (c1)\ v_{n+1}^{t+1} \neq 0 \tag{4-9}$$

False go decision (RLR)

$$v_n^t * t_Y < \mathrm{x}_\mathrm{n}^\mathrm{t} \tag{4-10}$$

Where,

t_Y is the yellow interval

$\mathrm{x}_\mathrm{n}^\mathrm{t}$ is the gap of the vehicle and the vehicle in front

In this research we denote the probability of occurrence of rear-end RS caused by stopped cars, non-stopped cars or slam on the brake as $P-RS1$, $P-RS2$ and $P-BRAKE$. The probability of occurrence of RLR is $P-RLR$.

7) Simulation Output

The simulation conducted based on C#. The output contains six documents.

The contents of the six documents are described as follow:

- Brake1-Contains data of emergency brake (BRAKE).

Each number in the Brake1 document represents the $P-BRAKE$ during each simulation process (Figure 4.4).

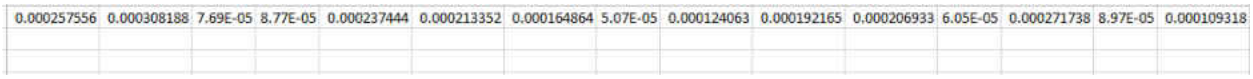
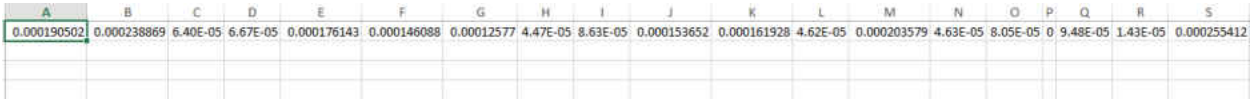


Figure 4.4 Output of Brake1 document

- DSZ1- Contains data of risky situations caused by stopped cars (RS1).

Each number in the Brake1 document represents the $P - RS1$ during each simulation process (Figure 4.5).

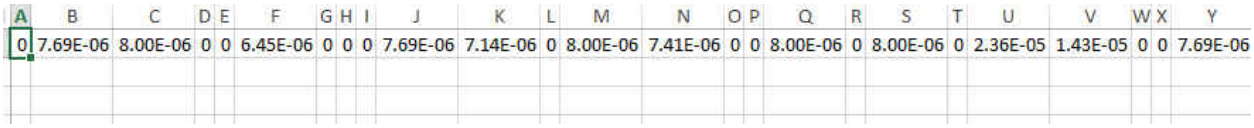


A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
0.000190502	0.000238869	6.40E-05	6.67E-05	0.000176143	0.000146088	0.00012577	4.47E-05	8.63E-05	0.000153652	0.000161928	4.62E-05	0.000203579	4.63E-05	8.05E-05	0	9.48E-05	1.43E-05	0.000255412

Figure 4.5 Output of DSZ1 document

- DSZ2- Contains data of risky situations caused by non-stopped cars (RS2).

Each number in the Brake1 document represents the $P - BRAKE$ during each simulation process (Figure 4.6).



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
0	7.69E-06	8.00E-06	0	0	6.45E-06	0	0	0	7.69E-06	7.14E-06	0	8.00E-06	7.41E-06	0	0	8.00E-06	0	8.00E-06	0	2.36E-05	1.43E-05	0	0	7.69E-06

Figure 4.6 Output of DSZ2 document

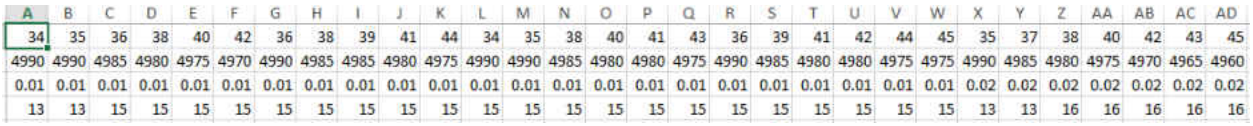
- tl1-Contains the spatial and temporal information of RS1 and RS2 (Figure 4.7).

The first line represents the time when the RS1 or RS2 happens;

The second line represents the location of the risky situation;

The third line records in which simulation process does the risky situation presents;

The last line shows the expected speed of the vehicles



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD
34	35	36	38	40	42	36	38	39	41	44	34	35	38	40	41	43	36	39	41	42	44	45	35	37	38	40	42	43	45
4990	4990	4985	4980	4975	4970	4990	4985	4985	4980	4975	4990	4990	4985	4980	4980	4975	4990	4985	4980	4980	4975	4975	4990	4985	4980	4975	4970	4965	4960
0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02
13	13	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	13	13	16	16	16	16	16

Figure 4.7 Output of tl1 document

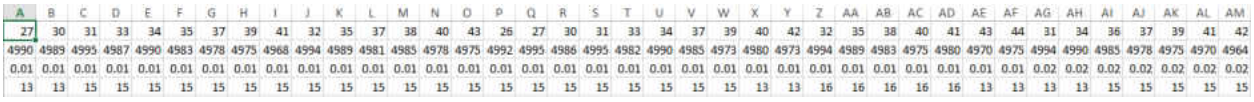
- tl2- Contains the spatial and temporal information of emergency brake (Figure 4.8).

  The first line represents the time when the emergency brake happens;

  The second line represents the location of the risky situation;

  The third line records in which simulation process does the risky situation presents;

  The last line shows the expected speed of the vehicles.

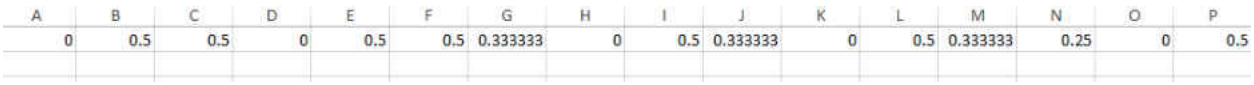


A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM
27	30	31	33	34	35	37	39	41	32	35	37	38	40	43	26	27	30	31	33	34	37	39	40	42	32	35	38	40	41	43	44	31	34	36	37	39	41	42
4990	4989	4995	4987	4990	4983	4978	4975	4968	4994	4989	4981	4985	4978	4975	4992	4995	4986	4995	4982	4990	4985	4973	4980	4973	4994	4989	4983	4975	4980	4970	4975	4994	4990	4985	4978	4975	4970	4964
0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02
13	13	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	13	13	16	16	16	16	16	13	13	13	13	15	15	15	15	15

Figure 4.8 Output of tl2 document

- stgo-error- Contains the information about drivers' false decisions $P - RLR$.

Each number in the Brake1 document represents the $P - RLR$ during each simulation process (Figure 4.9).



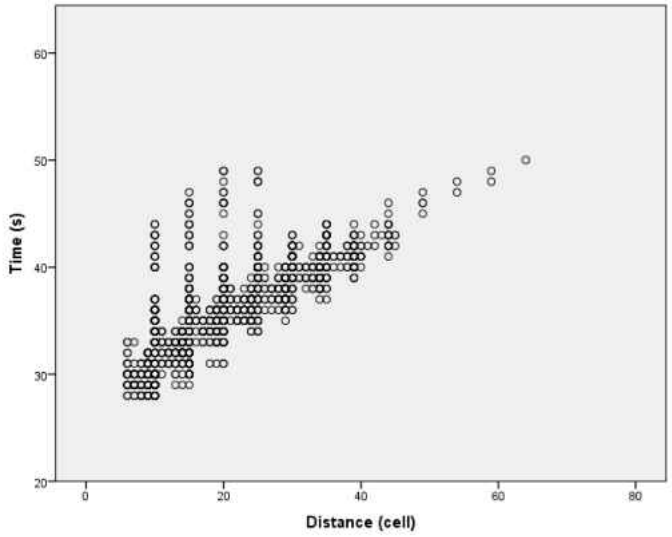
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
0	0.5	0.5	0	0.5	0.5	0.333333	0	0.5	0.333333	0	0.5	0.333333	0.25	0	0.5

Figure 4.9 Output of stgo-error document

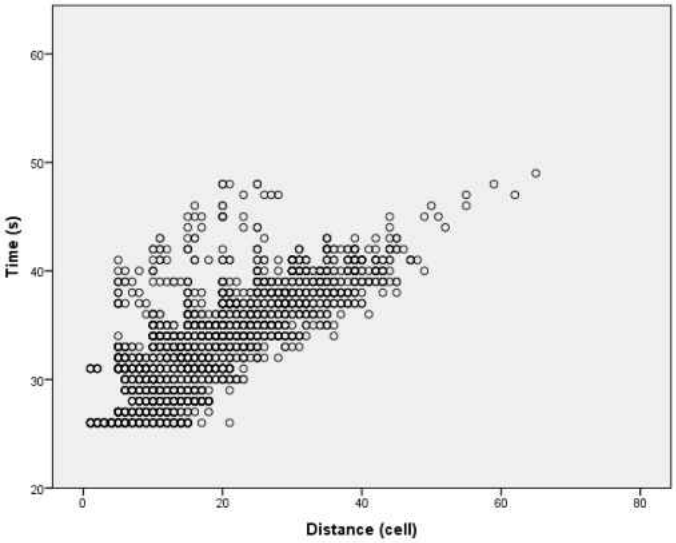
CHAPTER 5 SCENARIOS CONSTRUCTION AND ANALYSIS

5.1 Typical Intersection

A typical simulation will follow the general rules which are described in Chapter 4.2. The spatial and temporal information of the risky situations is depicted in Figure 5.1, when the expected mean speed of the leading vehicles follows the normal distribution N~ (50,5). Figure 5.1(a) shows the RS1 and RS2 risky situations, which represent the possible rear-end crashes. Most of this type of risky situation is present at the end of yellow phases and the beginning of red phases. One possible reason for these situations is the difference of the driver behavior during the yellow interval and their false judgments of other drivers' decisions (Yan et al., 2005). Meanwhile, most of the risky situations are located 10ft to 45ft away from the intersection. Figure 5.1(b) describes the distribution of the presence of the emergency brake. Different from the rear-end crash's risky situations, most of this type of risky situations are present closer to the intersection and begin to present soon after the onset of the yellow indication. Thus, most risky situations happen at the beginning of the yellow indication until 10 seconds after the onset of the red indication, and located at 7.5m (5cells) to 60m (40cells) away from the stop line.



(a)



(b)

Figure 5.1 Spatial and temporal distribution of risky situations at the typical intersection when the expected speed of lead vehicles follow N~(50,5)
(a) RS1 and RS2 (b) emergency brake

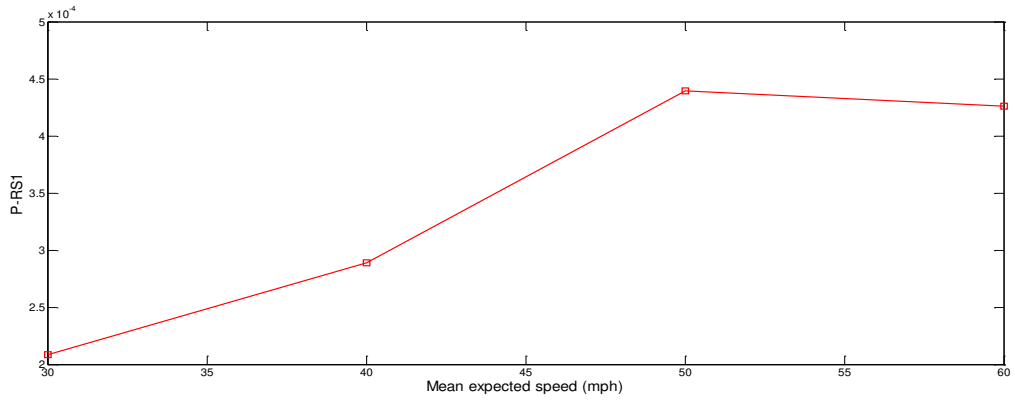
Figure 5.2 shows that the impact of increasing expected mean speed of leading vehicles on the presence of the risky situations. Significantly increasing trends of risky probabilities with the speed can be found. When the expected mean speed increased from 40mph to 50mph, the risky situations caused by stopped car will significantly increase, which indicates that many of the risky situations are due to the drivers' different stop/go decisions. When the expected mean

speed increased from 50mph to 60mph, the percentage of drivers who choose to stop will drop significantly because of the relatively larger parameter estimate of Speed Group 3 in the logistic model, which is equal to 3.2820. Thus, there is no significant increase of the risky situations caused by stopped cars (RS1) when the expected speed increased from 50mph to 60mph.

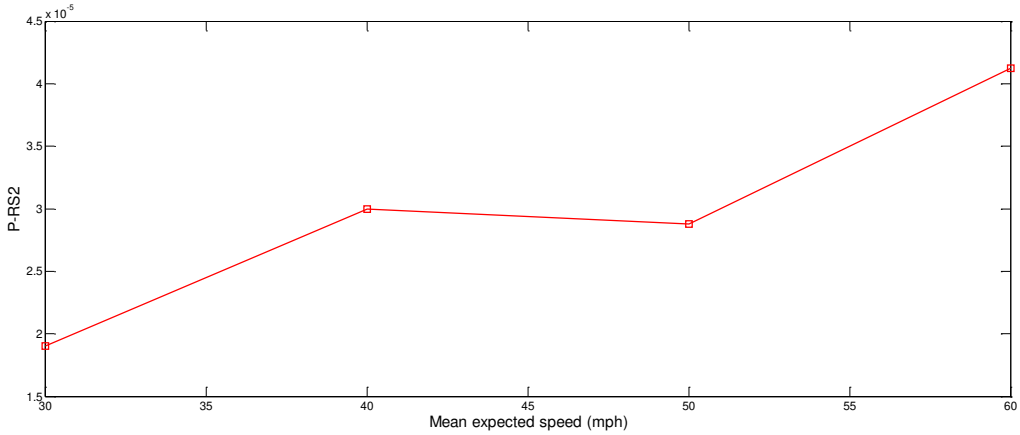
When increasing the standard deviation of the leading vehicles, the risk probabilities also show an increasing trend, especially during the standard deviation increased from 2mph to 5mph. If the standard deviation has already increased to 5mph, the probability of the risky situations will be more sensitive to the increase of the expected mean speed (Table 5.1).

Table 5.1 Impact of standard deviation of leading vehicles on risky situations at the typical intersection

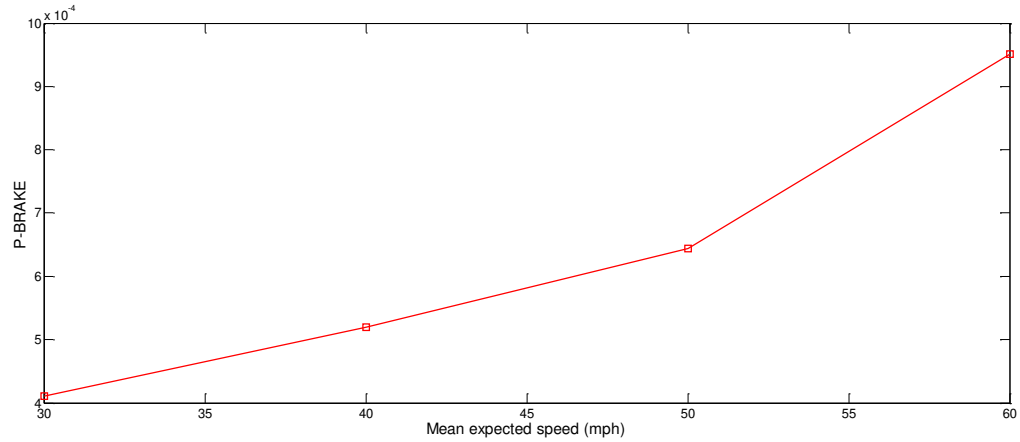
Speed Distribution / Risky Situation ($*10^{-5}$)	(50,2)	(50,5)	(50,10)
P-BRAKE	9.32	64.41	67.58
P-RS1	6.44	43.97	43.09
P-RS2	0.31	2.88	2.75



(a)



(b)



(c)

Figure 5.2 Impact of excepted mean speed of leading vehicles on Risky Situations at typical intersection
(standard deviations=5mph)
(a) P-RS1 (b) P-RS2 (c) P-BRAKE

Table 5.2 shows the potential RLR violations by different expected mean speed of the leading vehicles the typical intersection. When the expected mean speed lower than 50mph, there is an increasing trend of the percentage of false decisions with the increase of speed. If the expected mean speed increased from 50mph to 60mph, the percentage will be reduced. The definition of P-RLR is the percentage of false stop/go decisions of the drivers. From Chapter 3.2.2, vehicles traveling at a speed about 50mph are more prone to have the false go decision. Thus, the simulation scenarios that have more vehicles drive at this operating speed will have higher percentage of false decisions, and will lead to high P-RLR.

Table 5.2 RLR risk probabilities by different expected mean speed of leading vehicles at the typical intersection

Speed Distribution	(30,5)	(40,5)	(50,5)	(60,5)	(50,2)	(50,10)
RLR risk probability (*10^{-4})	37	56	166	52	323	134

5.2 Intersection with Flashing Green Signal

- Scenario Construction

In some countries, flashing green phases are implemented at the end of green phases to give drivers advanced warning for the upcoming yellow indication and it is still part of the green phases. Newton et al. (1997) found that about 80% of drivers make acceleration or deceleration decisions during the flashing phases in the Change Anticipation System (TLCAS) program. Many of them will decelerate, but some of the drivers will choose to accelerate. Also, the TLCAS maximum deceleration value (2.5m/s^2) is significantly different from the regular program (3.1 m/s^2), which does not include flashing indication phases. The maximum

acceleration is 1.6m/s^2 in the TLCAS program instead of 2.0 m/s^2 for the regular program. As for the reaction time, the mean value for the TLCAS is 2.05s, which is much larger than the regular program. Based on the results of the TLCAS program, the simulation process of intersection with the Flashing Green is shown in Figure 2. Four seconds before the onset of the yellow indication, the flashing green indication begins. To simplify the simulation, 75% of the drivers will decelerate with 2.0 m/s^2, 5% of the drivers will accelerate with 1.0 m/s^2,the other 20% of the drivers will approach the intersection with the same speed during the simulation. The default value for the flashing green reaction time in this simulation is 2 seconds. The rules of driver behavior after the onset of the yellow indication are same as the typical intersection (Figure 5.3).

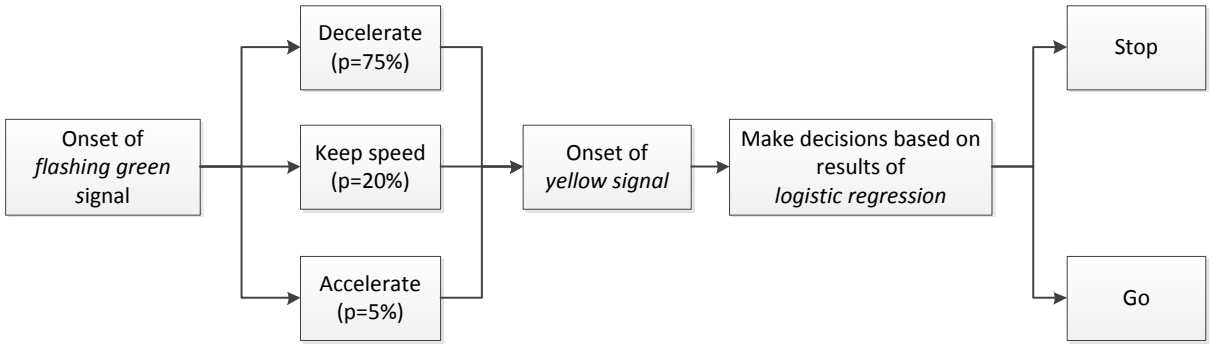
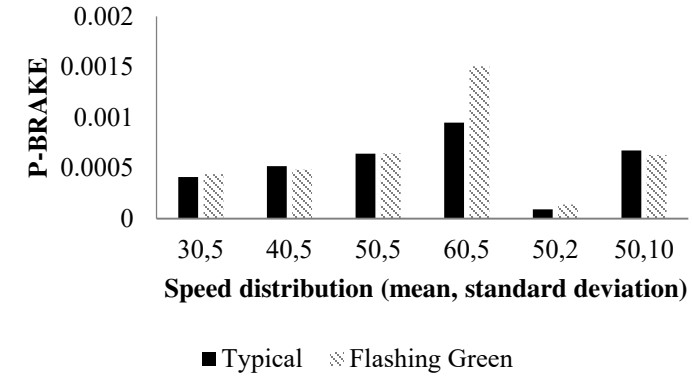


Figure 5.3 Simulation process of intersection with the flashing green indication

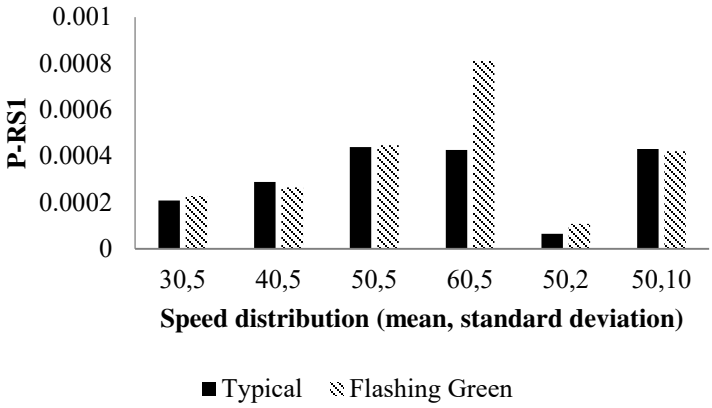
- Simulation Results

Figure 5.4 shows the comparative study of the intersection with the flashing green and the typical intersection. The flashing green phase leads to longer indecision interval of drivers, and the drivers' behaviors will be more various, which makes drivers harder to predict other drivers' behaviors and may lead to rear-end crashes. No significant improvement has been found. However, when the vehicles have high operating speed or low variance of the speed between

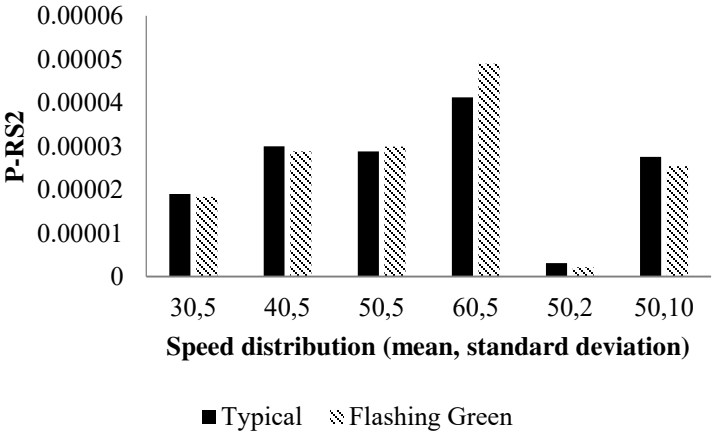
vehicles, relative higher risk probabilities have presented. The different acceleration or deceleration decisions during the flashing green phases are the reason for the increasing of the risky probabilities in some cases. The results are consistent with the results of the previous study (Factor et al., 2012)



(a)



(b)



(c)

x axis- (mean speed (mph), standard deviation(mph))

Figure 5.4 Impact of excepted mean speed of leading vehicles on risky situations at the typical intersection and the intersection with the flashing green

(a) P-RS1 (b) P-RS2 (c) P-BRAKE

Figure 5.5 shows the spatial and temporal information of the risky situations of the intersection with flashing green phases. When the expected mean speed is 30mph, most of the risky situations present during 25 seconds to 45 seconds with 7.5m (5 cells) to 60m (40 cells) away from the stop line. When the expected mean speed increased from 30mph to 60mph, the time range increase to 25 seconds to 50 seconds and the distance range increase to 7.5m (5 cells) to 75m (50 cells). Thus, both the time and the distance range of the risky situations' distributions were increased with the increase of the expected mean speed.

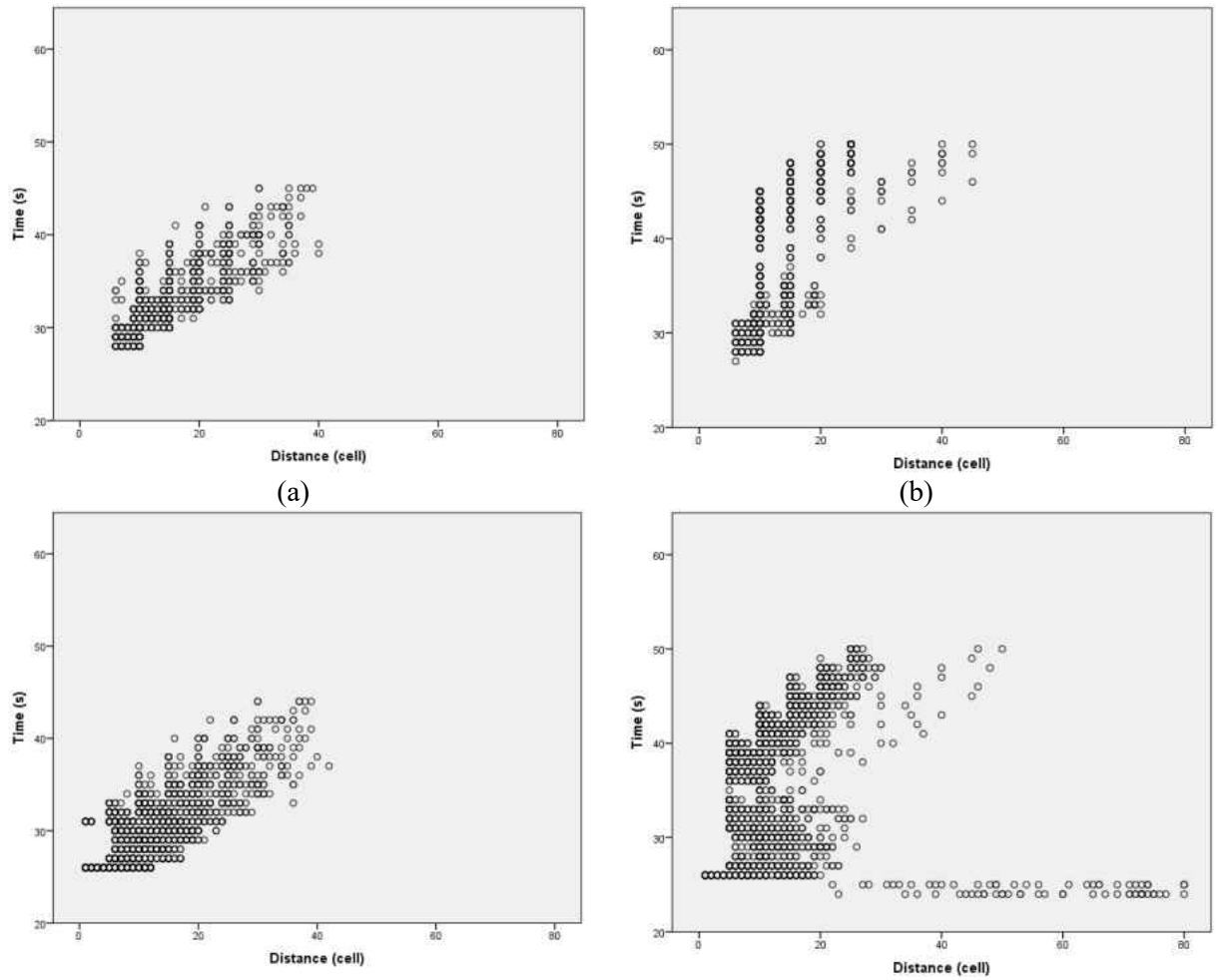


Figure 5.5 Spatial and temporal distribution of risky situations at the intersection with the flashing green phases
(standard deviation=5mph)

(a) RS1 and RS2 when the expected mean speed is 30mph (b) RS1 and RS2 when the expected mean speed is 60mph (c) emergency brake when the expected mean speed is 30mph (d) emergency brake when the expected mean speed is 60mph

The RLR risk of the flashing green does not have a significant difference compared to the typical intersection, because both of the drivers in the scenarios make their stop/go decisions based on the logistic model, which can be effect by speed, distance or the lead/follow position of the vehicles (Table 5.3).

Table 5.3 P-RLR of the typical intersection and the intersection with the flashing green

Scenario \ Expect Speed Distribution	(30,5)	(40,5)	(50,5)	(60,5)	(50,2)	(50,10)
Typical intersection scenario (*10^{-4})	37	56	166	52	323	134
Flashing green scenario (*10^{-4})	37	84	205	85	300	138

5.3 Intersection with Pavement Marking

- Scenario Construction

The rules of pavement marking is depicted in Figure 5.6, two cars (*A* & *B*) face the change of yellow indication. Since car ***B*** has passed the pavement marking, he/she should choose to go while the car **A** should choose to stop.

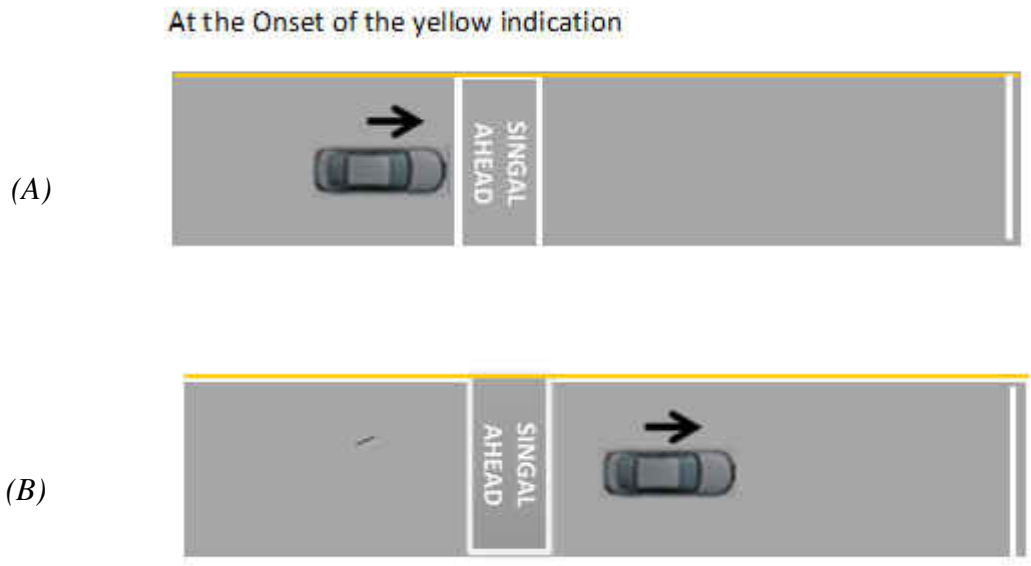


Figure 5.6 Scenario of intersection with pavement marking

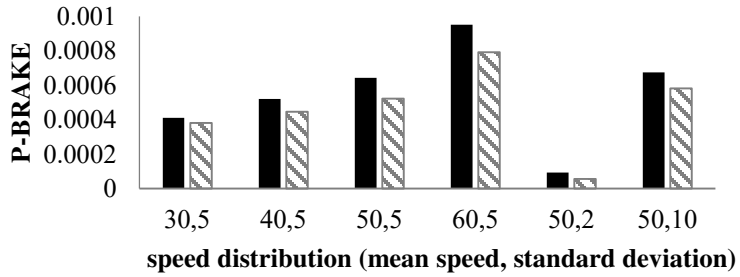
ITE's Engineering handbook suggests the distance from the marking to stop bar is (Pline, 1999),

$$X = V_0 t + \frac{{V_0}^2}{2a+19.6g} \quad (5\text{-}1)$$

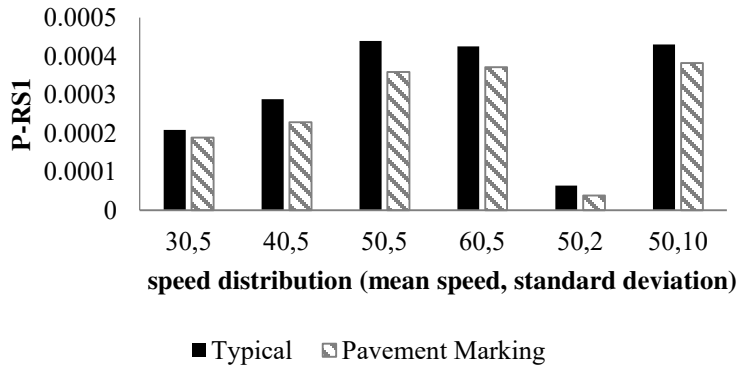
Where V_0 is the 85th percentile speed or speed limit; t is the reaction time; a is the average deceleration rate; g is the grade of the intersection. In this study, the pavement marking is designed with a 45 mph speed limit.

- Simulation Results

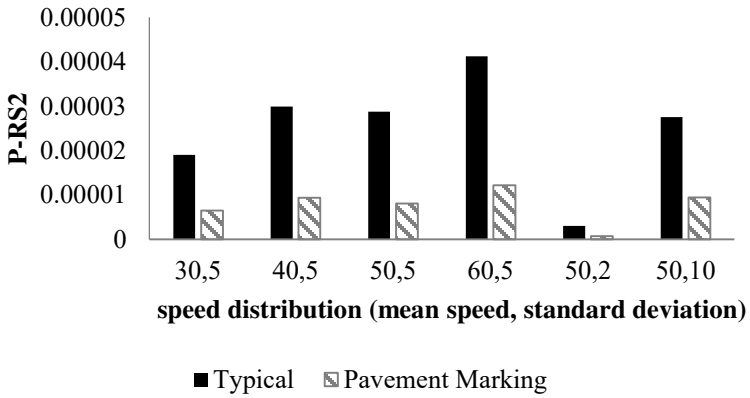
Figure 5.7 demonstrates the risky probabilities of pavement-marking scenarios. Significantly improvement can be observed of the three different types of risky situations, especially the risky situations caused by the non-stopped vehicles. The pavement marking countermeasures have more positive effect when the operating speeds of the vehicles are relatively higher. When the expected mean speed is 30mph, little effect can be found in reducing the risk probabilities. The probabilities of risky situations under the pavement-marking scenario increase with the increasing of the standard deviation and the expected mean speed.



(a)



(b)



(c)

Figure 5.7 Comparative risky probabilities analysis of the typical intersection and the intersection with pavement marking
(a) P-BRAKE (b) P-RS1 (c) P-RS2

Figure 5.8 depicted the presence of risky situation at the intersection with the pavement marking countermeasure. Compared with the simulation with the expected mean speed of 30mph and the simulation with 60mph, the pavement marking countermeasure can effectively prevent the expand of the distance range.

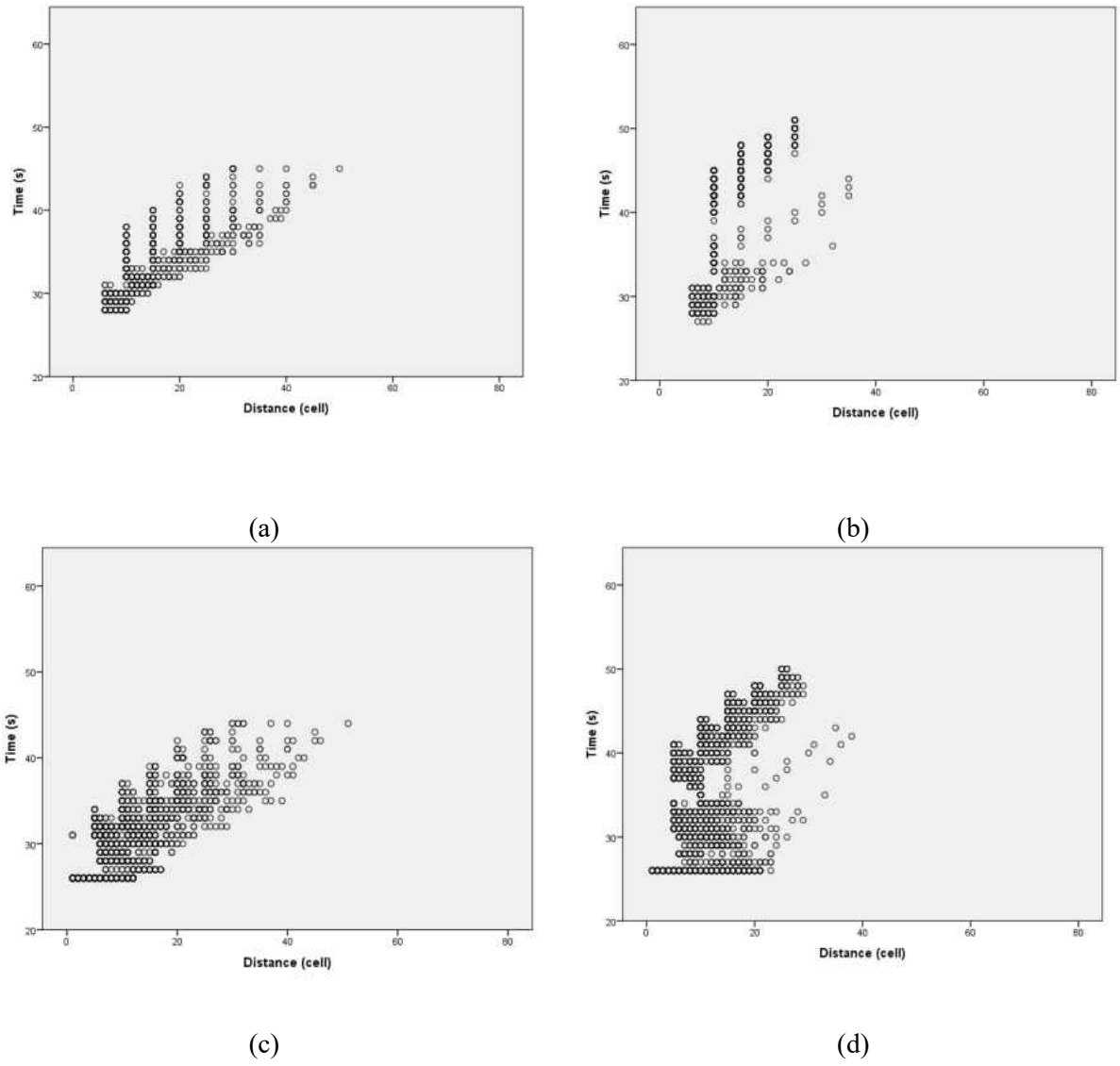


Figure 5.8 Spatial and temporal distribution of risky situations at the intersection with the pavement marking and an auxiliary indication countermeasure (standard deviation=5mph)

(a) RS1 and RS2 when the expected mean speed is 30mph (b) RS1 and RS2 when the expected mean speed is 60mph (c) emergency brake situations when the mean speed is 30mph (d) emergency brake situations when the expected mean speed is 60mph

However, the pavement marking significantly increase the risk of RLR violations when the vehicles with a low operating speed or a high variance. The reason for this phenomenon is that

the design speed for the pavement marking is 45mph. If the vehicle's speed is lower than the design speed, there will be high risk of red-light running violation. Thus, the pavement marking countermeasure have positive effects on reducing rear-end crashes risks. When the vehicles have low operating speed, the safety of the intersection will be dramatically decreased (Figure 5.9).

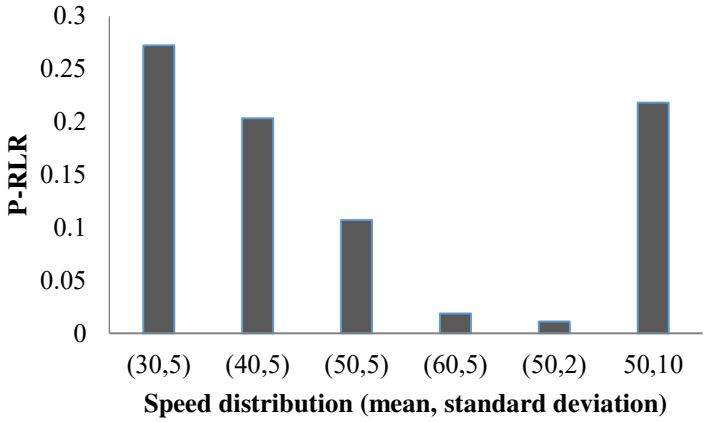
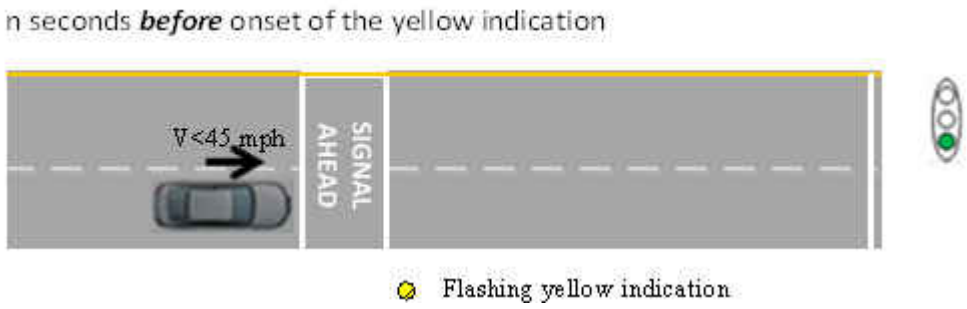


Figure 5.9 Probabilities of red-light running violation at the intersection with pavement marking

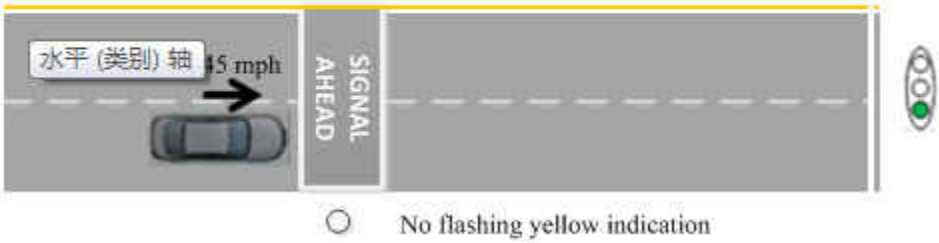
5.4 Intersection with Pavement Marking and an Auxiliary Indication countermeasure

- Scenario Construction

The flashing yellow signal beside the pavement marking is a warning signal, which will be onset if the speed of the vehicles is below the speed limit. The flashing yellow signal begins to flash a few seconds (about 1s to 3s) before the onset of the yellow indication and continues until the end of the red interval. Because of this disadvantage of pavement marking, a new countermeasure is proposed to solve this problem, which is installing an auxiliary flashing yellow signal next to the pavement marking. Thus, we propose a fourth scenario with both pavement marking and the auxiliary flashing yellow indication (PMAIC).



(a)



(b)

Figure 5.10 Scenario of the PMAIC
(a) approaching the intersection with a speed below the speed limit (b) approaching the intersection with a speed not below the speed limit

In Figure 5.10(a), when vehicles are approaching the intersection with a speed lower than the speed limit of the intersection approach (45mph), the auxiliary flashing yellow indication will begin flashing at *n* seconds before the yellow phase. If the vehicle that has not passed the pavement marking before the onset of the auxiliary flashing yellow indication and can see the flashing indication, the driver should choose to stop during the yellow interval. Otherwise, the driver should choose to go at the yellow duration. In Figure 5.10(b), when the operation speed of the vehicle is not below the speed limit (45mph), the driver should follow the rules that are similar to the pavement-marking scenario.

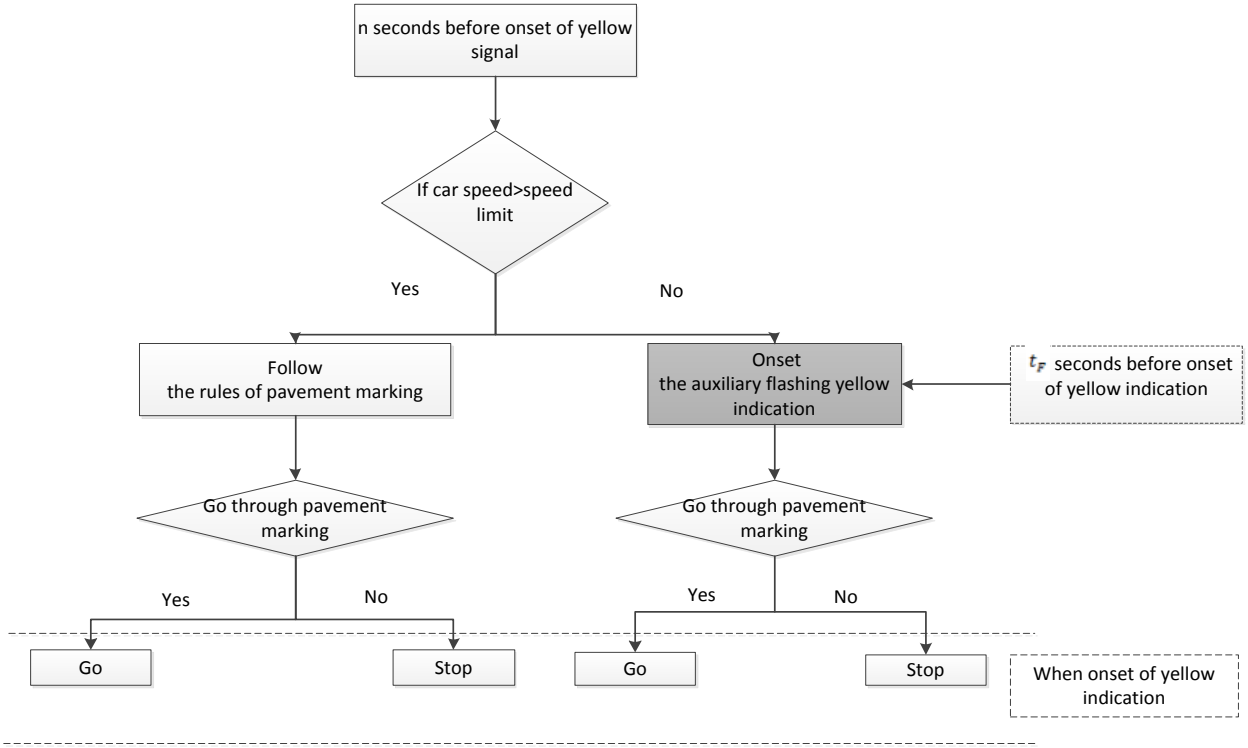


Figure 5.11 Simulation processes of pavement marking & auxiliary flashing indication

The processes of the PMAIC scenario simulation are shown in Figure 5.11. The value t_F is based on the vehicles' current speed at 5 seconds before the onset of the yellow indication.

$$V_c t_F = V_0 t_Y - V_c t_Y \tag{5-2}$$

Then,

$$t_F = \frac{V_0 t_Y - V_c t_Y}{V_c} \tag{5-3}$$

Where,

V_0 denotes the speed limit or 85th percentile speed

V_c denotes speed when n seconds before the onset of the yellow indication

t_Y denotes the yellow interval

Also, the value for the judgment time n is,

$$n = \frac{V_0 t_Y - V_{min} t_Y}{V_{min}} + 1 \quad (5\text{-}4)$$

Where,

V_{min} denotes the expected minimum speed of vehicles
For example, if a vehicle approaches the intersection with 30mph (13.4m/s), which is lower than the speed limit 45 mph (20m/s). Then 2.3 seconds before the yellow phase, the auxiliary flashing yellow indication next to the pavement marking will begin to flash.

- Simulation Results

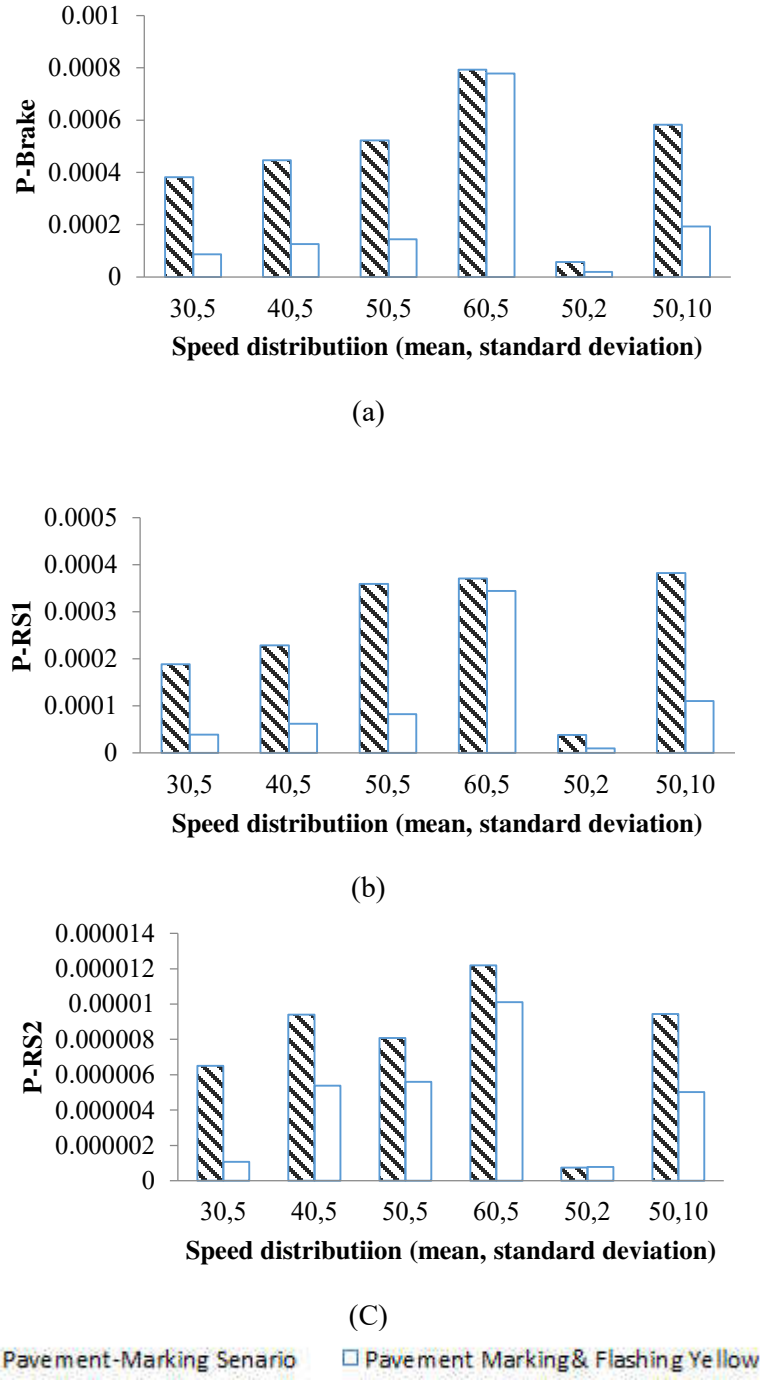
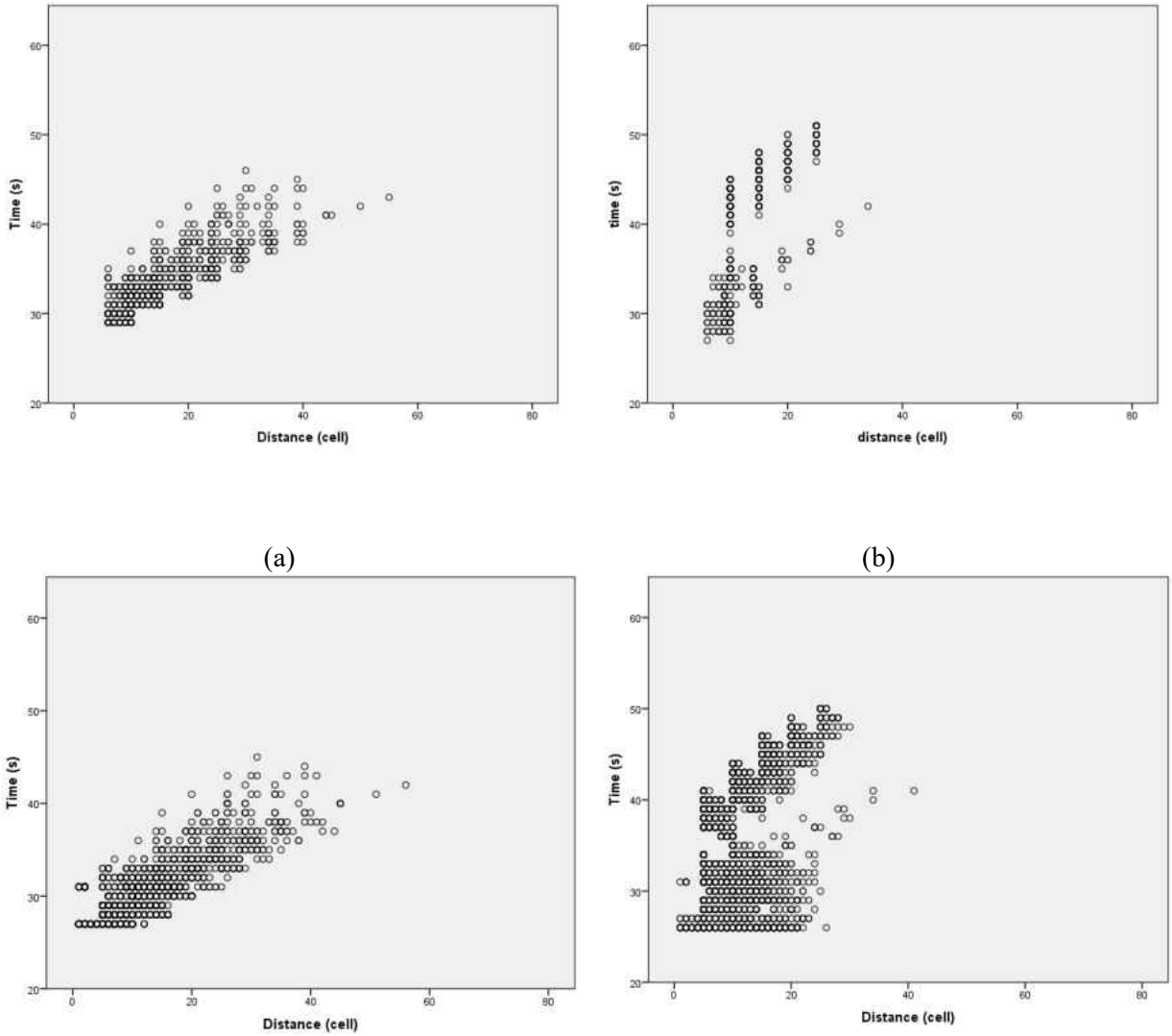


Figure 5.12 The probabilities of different types of rear-end RS under pavement-marking scenario and the PMAIC scenario
(a) RS1 (b) RS2 (c) BRAKE

The PAMIC scenario simulation results are shown in Figure 5.12. Compared with the pavement marking scenarios, the intersection with the PMAIC has less rear-end risk situations. The decrease of probability of rear-end RS under the PMAIC becomes much less apparently when the average speed is above 45mph, because the cars will follow the rules of pavement marking if their speeds are larger than 45mph. Like other scenarios, the probability of risky situations under the PMAIC scenario increases with the standard deviation of speed. However, compared with other scenarios, it can also decrease the probability of rear-end crash risks under the situation of high variance operating speed.

Figure 5.13 demonstrates the risky situations with different expected mean speed. The number of risky situations will increase with the increasing of expected mean speed. Similar with the pavement marking countermeasure, the PMAIC can also effectively prevent the increase of the distance range.



(a) (b)

(c) (d)

Figure 5.13 Spatial and temporal distribution of risky situations at the intersection with the PMAIC (standard deviation=5mph)

(a) RS1 and RS2 when the expected mean speed is 30mph (b) RS1 and RS2 when the expected mean speed is 60mph (c) emergency brake situations when the mean speed is 30mph (d) emergency brake situations when the expected mean speed is 60mph

In addition, during the new-countermeasure scenario simulation, rare RLR violations happen. Therefore, the PMAIC can effectively reduce both the rear end risky situations and the RLR violation.

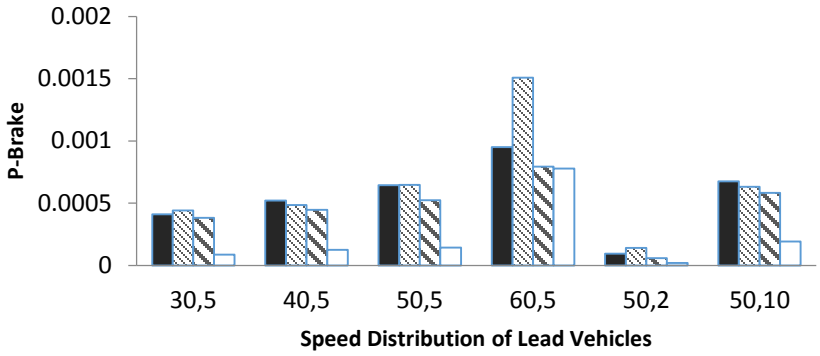
CHAPTER 6 COMPARATIVE ANALYSIS OF DIFFERENT DILEMMA ZONE COUNTERMEASURES

6.1 Rear-end Crash Risk

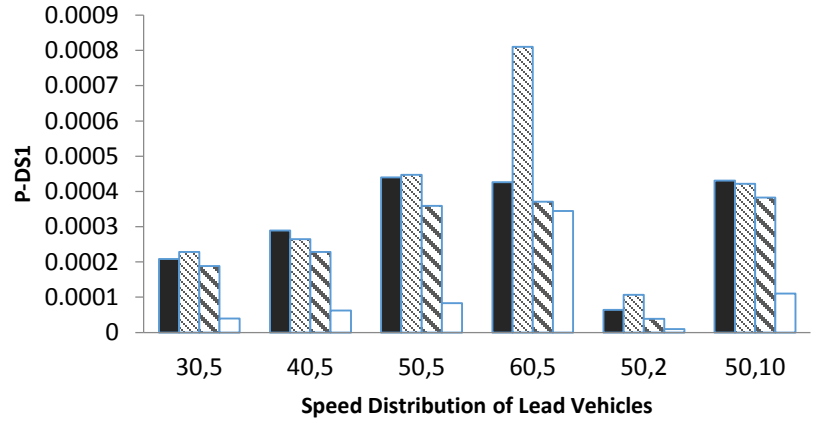
Figure 6.1 demonstrates the rear-end crash risks of the different intersection scenarios. A clear trend can be found: the mean speed or standard deviation can influence the BRAKE & RS2 risk probabilities. The RS1 risks for scenarios do not increase obviously when the speed rises from 50mph to 60mph. Low standard deviation shows a positive effect on safety improvement. In all of the four scenarios, the (50, 2) speed distribution simulations have the lowest risk probabilities. The results reveal that the main contributing causes to accident risk are high mean speed and high standard deviation of the speed distribution. Figure 4 also illustrates that the standard deviation will not have a significant impact on rear-end RS when it is greater than 5mph.

From the comparison, the results also show that the flashing green countermeasure does not improve safety significantly, especially under the situations of a high mean speed or a low standard deviation of speed distribution. The rear-end risk probabilities of the flashing green countermeasure are even higher than the probabilities of the typical intersection. Distinction between the typical intersection scenario and the flashing green scenario is probably due to the increase of the indecision period when drivers behave differently. Even though previous studies found that many drivers will decelerate during the flashing green interval and will somehow decrease the probability of rear-end crash risky situations, some of the flashing green intersections still suffer from high-risk probabilities for all of the three types of risk situations. In 2004, Köll et al. (2014) found that the flashing green phases is associated with a substantial increase of early stop. However, it also produces a larger indecision zones and lead to longer

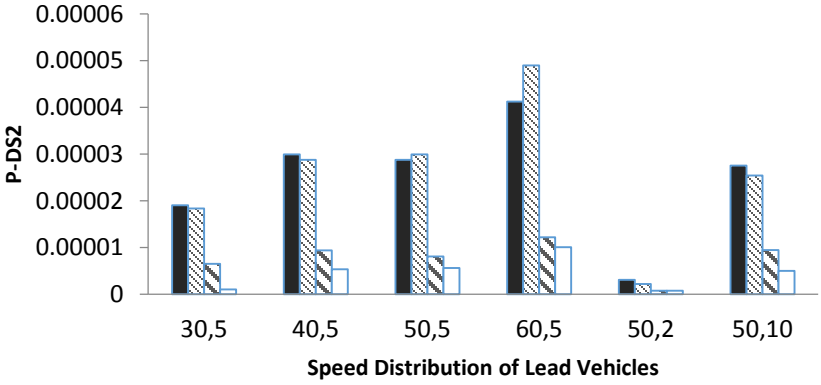
period of uncertainty, where following drivers cannot easily predict the front vehicles' stop/go decision (Factor et al., 2012). With respect to the pavement-marking scenario, it can decrease the rear-end crash risks, all of the risk probabilities of the intersection with pavement marking are lower than the typical intersection and the intersection with flashing green phases especially the rear-end crashes caused by non-stopped cars, which means the front vehicles in the crashes chose to cross the intersection during the yellow interval. The PMAIC can effectively reduce the rear-end crash risk probabilities especially at the low expected mean speed as well as the scenario with a high standard deviation.



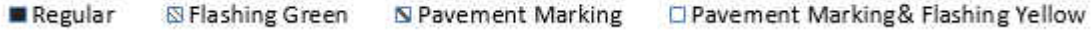
(a)



(b)



(c)



x-axis (expected mean speed (mph), standard deviation (mph))

Figure 6.1 The probability of different kinds of rear-end RS under different scenarios

(a) RS1 (b) RS2 (c) BRAKE under the scenarios of typical intersection, the intersection with flashing green and the intersection with pavement marking

Compared with the typical intersection scenario, the flashing green scenario have larger range of the emergency brake risk of both time and distance distribution. Because of the difference of the driver behavior during the flashing green phases, there are more risky situations present at the flashing green interval (Figure 6.2).

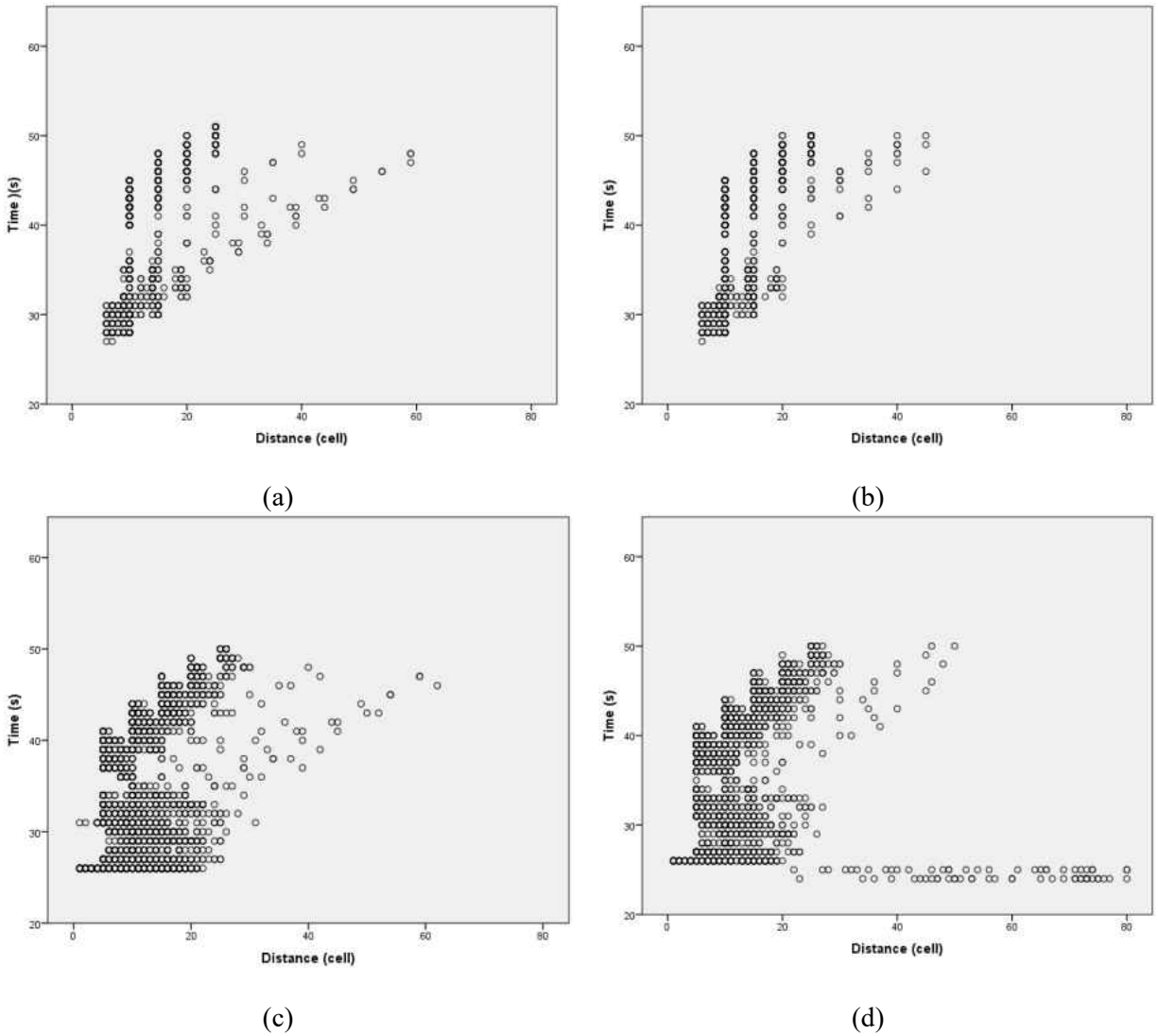


Figure 6.2 Spatial and temporal distribution of risky situations when the mean excepted speed is 60mph
(standard deviation=5mph)
(a) RS1 and RS2 of the typical intersection scenario (b) RS1 and RS2 of the flashing green scenario (c) emergency brake situations of the typical intersection scenario (d) emergency brake situations of the flashing green scenario

Compared with the spatial and temporal distribution of pavement marking, fewer risky situations can be observed at the PMAIC scenario, especially when during the yellow interval and the beginning of the red interval. The results indicate the PMAIC can effectively mitigating the problem of the dilemma zone (Figure 6.3).

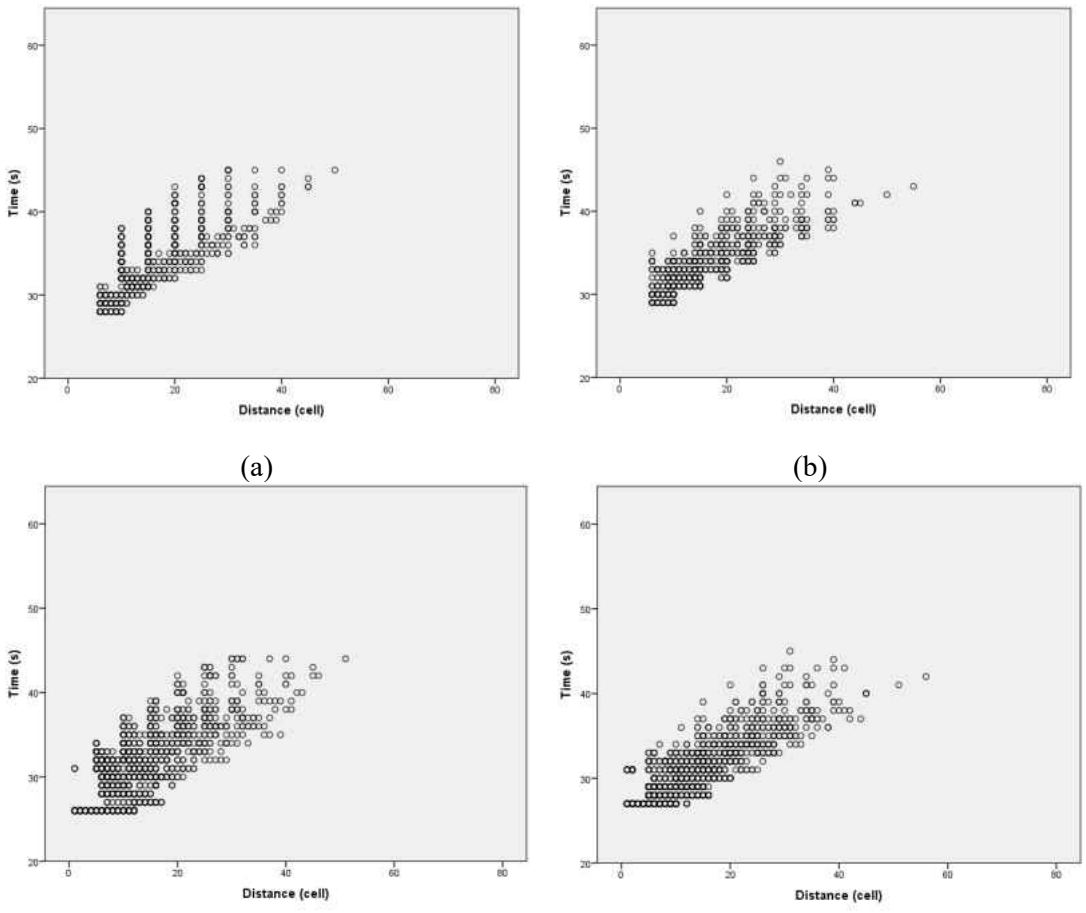


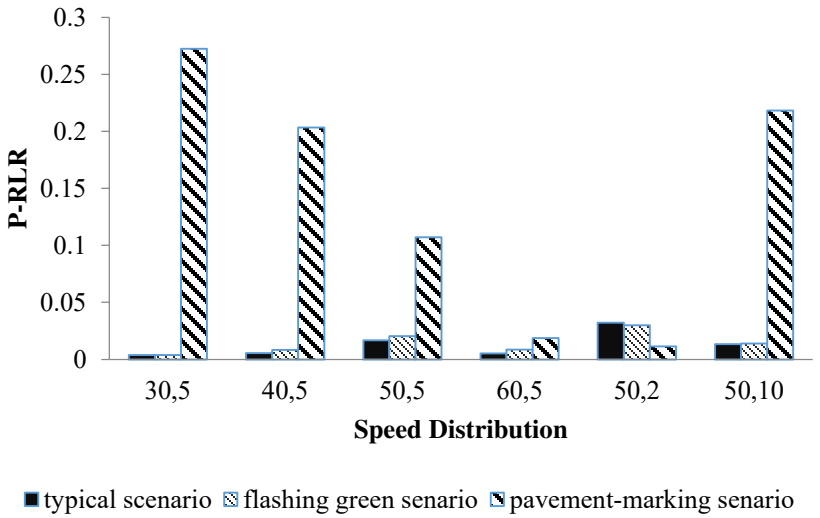
Figure 6.3 Spatial and temporal distribution of risky situations when the mean excepted speed is 30mph
(standard deviation=5mph)

(b) RS1 and RS2 of the pavement marking scenario (b) RS1 and RS2 of the PMAIC scenario (c) emergency brake situations of the pavement marking scenario (d) emergency brake situations of the PMAIC scenario

6.2 Red –Light Running Risk

At the yellow duration, drivers' false stop/go decisions can lead to red-light running (RLR) violations. During the simulation, the percentage of false decisions, which is the potential red-light running violations, is calculated. The results are shown in Figure 6.4. In the scenarios of typical intersection signal and the intersection with the flashing green phases, comparison between different lead vehicles' speed distribution illustrates that the mean speed is an important factor for the decision-making. The highest risk probabilities occur at 50mph mean speed distribution. It can also be found that the standard deviation has little impact on the drivers' typical stop/go decision.

It is pointed out in previous studies that the drivers would be more prone to stop at the intersection with the flashing green/yellow phases (Newton et al., 1997), This is because that many of the drivers will decelerate during the flashing green phases and lead to lower speed at the onset of the yellow indication. These driver behaviors will increase the stop decisions according to the logistic regression model of stop/go decisions. However, drivers will still make decisions based on their own judgment, which means the percentage of false go decisions will not decrease. The simulation results suggest that the flashing green phase measure cannot effectively decrease the percentage of false decisions by drivers.



x-axis (Mean Speed (mph), Standard Deviation (mph))

Figure 6.4 the probability of RLR RS under different scenarios

From the analysis of rear-end crash risk, it seems that the pavement marking is an effective countermeasure to improve rear-end risk. However, Figure 6.4 demonstrates that the RLR violation is significant when the mean speed of the leading vehicles' speed distribution is lower than 50mph or the standard deviation of the speed distribution is high. The disadvantage of the pavement marking is that if a driver encounters the yellow signal at a speed lower than the speed limit, even though he/she has passed the pavement marking, there is still a high chance for RLR and he/she cannot execute the go maneuver safely. Such a negative situation may result in red-light running due to the lower approaching speed. Therefore, the pavement marking can effectively improve the intersection's safety only when the vehicles are approaching the intersection with high speed and low speed differences between vehicles. Otherwise, the pavement marking countermeasure can also lead to high chance of RLR violations. Rear RLR violations happen during the simulation of the PMAIC scenario.

CHAPTER 7 CONCLUSIONS

7.1 Research Contributions

Driver behavior during the yellow interval is influenced by the operating speed, the distance to the stop line, and the lead or follow position of the vehicle in a platoon. Vehicles with a higher speed, closer to the stop line or follow position vehicles are more prone to cross the intersection. A logistic regression model was used to predict the stop/go decision of drivers as a function of distance to the stop line, the operating speed and the lead/follow position. Most of the RLR violations are caused by drivers' false stop/go decisions during the yellow interval. The three variables, which include speed, distance and lead/follow position, have a significant impact on the RLR violations.

A logistic model can be employed to predict the drivers' stop/go decisions during the yellow interval. Vehicles with a larger distance from the stop line, a lower speed or a lead position are more prone to stop during the yellow interval.

From the simulation results, the mean speed and the standard deviation play a significant role in rear-end crash risk situations, where a lower speed and lower standard deviation could lead to less rear-end crash risk situations at the same intersection. High differences in speed are more prone to cause rear-end crashes.

According to the comparative analysis (Table 7.1), the flashing green countermeasure has little influence on rear-end crash risk reduction. The difference between drivers' deceleration or acceleration decisions might be the major reason for the presence of the accident risk in the flashing green scenarios. Meanwhile, the pavement marking countermeasure can effectively decrease the rear-end crash risk in most situations, especially the rear-end crashes caused by non-

stopped cars, which means the front vehicles in the crashes chose to cross the intersection during the yellow interval. The PMAIC, which is adding an auxiliary flashing yellow indication next to the pavement marking, can further reduce the rear-end crash risks when the expected mean speed of the leading vehicles is relatively low.

With respect to the RLR violations, the RLR risk analysis shows that the mean speed of the leading vehicles has important influence on RLR risk~~s~~ in the typical intersection simulation scenarios as well as intersections with flashing green phases simulation scenarios. The results indicate that the flashing green phases cannot reduce the percentage of false go decisions, because the drivers make the stop/go decisions based on their own speed and position instead of the other drivers' approaching speeds. The pavement marking can effectively reduce the RLR risk situations when the vehicles are approaching the intersection with high speed and low speed differences with other vehicles. Otherwise, the intersection will suffer from a high potential of RLR violation. The PAMIC has rare RLR violations. Therefore, PAMIC can effectively improve safety at signalized intersections.

Table 7.1 Comparative analysis of different dilemma zone countermeasures

	Flashing Green Countermeasure	Pavement Marking Countermeasure	PMAIC
Rear-end crash risk	Increase the risk of rear-end crash	Decrease the risk of rear-end crash	Decrease the risk of rear-end crash, especially when the expected mean speed is relatively lower
RLR violations	No significant impact on reducing RLR risk	Significantly increase the RLR with low expected mean speed	Effectively reduce the RLR violations' probabilities

Spatial and temporal analyses of the risky situations indicate that both the distance and the time range of the risky situations will be increased with the increase of the operating speed of the vehicles. The pavement marking countermeasure and the PMAIC can effectively prevent the increase of the distance range with the increase of the operating speed. Comparative analyses of different scenarios demonstrate the effectiveness of the PMAIC in reducing the probabilities of risky situations.

7.2 Recommendations

The simulation and the conclusions of this research effort have made several recommendations for further enhancement and improvement.

- More field data collection or driving simulator experiments of the driver behavior at the intersections with different countermeasures could be conducted to analyze the real driver response to the countermeasures.
- Further simulation could be conducted for the comparative analyses. In this study, each simulation scenario only contains one lane. Multi-lane scenarios could be constructed. Also, other countermeasures could be tested in the CA developed simulation.
- Further research could be developed for the PMAIC. A multi-lane scenario simulation and driving simulator experiments are recommend. Meanwhile, the fluctuation and the difference of the operating speed should also be considered. In the future, similar solutions under the Vehicle to Infrastructure Integration (VII) initiative could be adopted by replacing the external yellow indication with a sound or indication inside the car.
- Although the CA model has its advantage in traffic flow simulation, few software can apply the CA model. User-friendly software that uses the CA model to simulate the traffic flow is recommended.

APPENDIX A C# CODE FOR THE TYPICAL INTERSECTION SCENARIO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
namespace No_Countermeasure
{
    class Program
    {
        Random random = new Random();
        static void Main(string[] args)
        {
            Random random = new Random();
            int Road_Length = 5000;
            int Vmax = 20;
            int Vfast = 20;
            double[] P = { 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 };
            int t0 = 0;
            int TT1 = 10000;
            int TT = 5000;
            double alpha;
            int L_veh = 5;
            int green = 25;
            int C = 60;
            int red = 25;
            int ki;
            int kii = 0;
            int error = 0;
            int D = 2;
            int t_safe = 3;
            int Vlimit = 15;
            double[,] t_l1 = new double[4, 5000];
            double[,] t_l2 = new double[4, 5000];
            int t_l11 = 0;
            int t_l22 = 0;
            double[] DSZ11 = new double[100];
            double[] DSZ22 = new double[100];
            int[] Flux11 = new int[100];
            double[] Slam_Brake11 = new double[100];
            double[] stgo_error = new double[10000];
            int stgo_para = 0;
            int CC = 1;
            int k = 0;
            for (int mm = 0; mm < CC; mm++)
            {
                double[] DSZ1 = new double[100];
                double[] DSZ2 = new double[100];
                int[] Flux = new int[100];
                double[] Slam_Brake = new double[100];
                int[,] Time_Location = new int[300, Road_Length];
                int[,] ArrTL = new int[240, 2000];

                int M = 0;
                int Dis = 0;
                int para_signal = 0;
                Program pm = new Program();
                for (alpha = 0.01; alpha < 0.401; alpha += 0.01)
                {
```

```
#region
int[] Location = new int[Road_Length];
int[] Velocity = new int[Road_Length];
double[] DS1 = new double[TT];
double[] DS2 = new double[TT];
double[] Slam_Brake1 = new double[TT];
int[] Desired_speed = new int[5000];
int[] Desired_speed_1 = new int[5000];
int[] Velocity_1 = new int[Road_Length];
int[] stgo_1 = new int[Road_Length];
int[] Velocity_2 = new int[Road_Length];
int[] stgo = new int[5000];
int signal_control = 1;
int tt = 0;
#endregion
for (int t = t0; t < (TT1 + TT); t++)
{
    if (tt == green + 5)
    {
        for (int ii = 0; ii < Road_Length; ii++)
        {
            stgo[ii] = 0;
        }
    }
    #region
    if (t > t0)
    {
        if (tt >= C)
        {
            tt = tt - C;
        }
        tt = tt + 1;
        if (tt <= green)
        {
            signal_control = 1;
        }
        else
        {
            if (tt <= green + 4)
            {
                signal_control = 2;
            }
            else
            {
                signal_control = 3;
            }
        }
    }
    #endregion
    #region
    if (t % 60 == 0)
    {
        double cs = random.Next(3, 6);
        int born_number = (int)(Math.Floor(cs));
        int dspeed = pm.NormalRandom();
        ki = born_number;
        for (int ii = 0; ii < born_number; ii++)
        {
```

```
            Location[ii * (dspeed + L_veh)] = 1;
            Velocity[ii * (dspeed + L_veh)] = dspeed;
            Desired_speed[ii * (dspeed + L_veh)] = dspeed;
        }
    }
    else
        ki = 0;
    #endregion

    #region

    int N = 0;
    for (int j = 0; j < Road_Length; j++)
    {
        if (Location[j] == 1)
        {
            N++;
        }
    }
    if (N == 0)
    {
        continue;
    }
    if (t == 3000)
    {
        t = 3000;

    }
    int[] Location_1 = new int[N];
    int[] Num_Location_1 = new int[N];
#endregion

    #region
    if (tt > green && tt < green + 5)
    {
        for (int j = 0; j < Road_Length; j++)
        {
            stgo_1[j] = stgo[j];
        }
        for (int j = 0; j < Road_Length; j++)
        {
            stgo[j] = 0;
        }
        for (int j = 0; j < Road_Length - 1; j++)
        {
            stgo[j + ki] = stgo_1[j];
        }
    }
    #endregion
    #region

    int[] Distance = new int[N];
    int[] Num_Location = new int[N];
    int[] Num_Velocity = new int[N];
    int[] Virtual_Velocity = new int[N];
    int[] kkk = new int[N];
    int Number = 1;
    int i = 0;
```

```
while (i < Road_Length)
{
    int[] a = new int[2];
    int c = 0;
    while (i < Road_Length && c < 2)
    {
        if (Location[i] == 1)
        {
            a[c] = i;
            c++;
        }
        i++;
    }
    if (c == 2)
    {
        i--;
        c = 0;
        Distance[Number - 1] = a[1] - a[0] - L_veh;
        Num_Location[Number - 1] = a[0];
        Num_Velocity[Number - 1] = Velocity[a[0]];
        Number++;
    }
    else
    {
        if (a[0] >= 4995)
        {
            Distance[Number - 1] = Vmax;
            switch (signal_control)
            {
                case 1:
                    Distance[Number - 1] = Vmax;
                    break;
                case 2:
                    if (stgo[Number - 1] == 1)
                        Distance[Number - 1] = Vmax;
                    else
                        Distance[Number - 1] = 0;
                    break;
                case 3:
                    if (tt <= green + 5)
                        Distance[Number - 1] = Vmax;
                    else
                        Distance[Number - 1] = 0;
                    break;
            }
        }
        else
        {
            if (signal_control == 1)
            {
                Distance[Number - 1] = Vmax;
            }
            else
            {
                if (signal_control == 2)
                {
                    if (stgo[Number - 1] == 1)
                        Distance[Number - 1] = Vmax;
```

```
                    else
                Distance[Number - 1] = Road_Length - a[0] - L_veh;
                                                                }
                 else
                 {
               Distance[Number - 1] = Road_Length - a[0] - L_veh;
                 }

            }
        }
        Num_Location[Number - 1] = a[0];
        Num_Velocity[Number - 1] = Velocity[a[0]];
        break;
    }
}
for (int j = 0; j < Number; j++)
{
    if (Distance[j] < 0)
    {
        error++;
    }
}
#endregion

for (int j = 0; j < Number; j++)
{
    if (Distance[j] < 0)
    {
        error++;
    }
}

for (int j = 0; j < Number - 1; j++)
{
    if (Num_Velocity[j] - Num_Velocity[j + 1] > Distance[j])
    {
        error++;
    }
}



#region
if (t > TT1)
{
    for (int j = ki + kii; j < Number - 1; j++)
    {
        if (ki > 0)
            error++;
        if (kii > 0)
            error++;

        if (Num_Velocity[j] < Vmax && Distance[j] == 0)
        {
            if (Velocity_2[j + 1 - ki - kii] >= 0)
            {
                if (Velocity_2[j - ki - kii] - Num_Velocity[j] >
2)
```

```
                                    {

                                        if (Num_Velocity[j + 1] == 0)
                                        {
                                            DS1[t - TT1 - 1] += 1;
                                        }
                                        else
                                        {
                                            DS2[t - TT1 - 1] += 1;
                                        }
                                        if (t_l11 < 5000)
                                        {
                                            t_l1[0, t_l11] = tt;
                                            t_l1[1, t_l11] = Num_Location[j];
                                            t_l1[2, t_l11] = alpha;
                                            t_l1[3, t_l11] =
Desired_speed[Num_Location[j]];

                                            t_l11++;
                                        }
                                    }
                                }
                            }

                        }
                        DS1[t - TT1 - 1] /= (Number);
                        DS2[t - TT1 - 1] /= (Number);
                    }
                    #endregion

                    Program pp = new Program();

                    #region
                    if (tt == green + 2)
                    {
                        for (int j = 0; j < Number; j++)
                        {
                            if (Num_Location[j] > 4900 && Num_Location[j] < 4995)
                            {
                                double sj = random.Next(0, 999);
                                sj /= 1000;
                                bool lead = false;
                                if (j == Number - 1)
                                    lead = true;
                                double jj = pp.ST_GO_LogitRegression(lead,
Num_Velocity[j], 4995 - Num_Location[j]);
                                if (jj > 0.48)
                                    stgo[j] = 1;
                                else
                                    stgo[j] = 2;


                            }
                            else
                            {
                                stgo[j] = 2;
                            }
                            int d2 = 0;
                            if (j == Number - 1)
```

```
                                {
                                    if (Num_Velocity[j] % 2 == 0)
                                        d2 = Num_Velocity[j] * (Num_Velocity[j] - 2) / 4;
                                    else
                                      d2 = (Num_Velocity[j] - 1) * (Num_Velocity[j] - 1) / 4;
                                    if (d2 < 4995 - Num_Location[j])
                                        stgo[j] = 1;
                                }
                                else
                                {
                                    stgo[j] = 2;
                                }
                if (4995 - Num_Location[j] <= 2 * Num_Velocity[j] && Num_Velocity[j] > 12)
                                    stgo[j] = 1;
                                if (Num_Velocity[j] >= 13 && Num_Location[j] >= 4960)
                                    stgo[j] = 1;

                                if (4995 - Num_Location[j] > 4 * Num_Velocity[j])
                                {
                                    stgo[j] = 2;
                                }
                            }
                            if (Number >= 3)
                            {
                                for (int j = Number - 2; Num_Location[j] > 4920 && j > 1;
j--)
                                {
                                    if (Num_Velocity[j] > Distance[j])
                                        stgo[j] = stgo[j + 1];
                                }
                            }
                        }
                        if (tt > green + 2 && tt < green + 4)
                        {
                            for (int j = 0; j < Number; j++)
                            {
                    if (Num_Location[j] > 4920 && Num_Location[j] - Num_Velocity[j] <=
4920)
                                    stgo[j] = 2;
                            }
                        }
                        if (tt == green + 2)
                        {
                            for (int j = Number - 1; stgo[j] > 0 && j > 0; j--)
                            {
                                if (stgo[j] == 2 && j > 0)
                                {
                                    for (int jj = j; stgo[jj] > 0 && jj > 0; jj--)
                                    {
                                        stgo[jj] = 2;
                                    }
                                    break;
                                }
                            }
                        }

                        #endregion
                        #region
```

```
                        if (tt == green + 2)
                        {
                            int Num_stgo_error = 0;
                            int Num_pdcar = 0;
                            for (int j = Number - 1; Num_Location[j] > 4900; j--)
                            {
                                Num_pdcar++;
                                if (stgo[j] == 1 && ((5000 - Num_Location[j]) > 4 *
Num_Velocity[j]))
                                    Num_stgo_error++;
                            }
                            if (Num_pdcar > 0)
                            {
                                stgo_error[stgo_para] = (double)Num_stgo_error /
Num_pdcar;
                                stgo_para++;
                            }
                        }
                        # endregion

                        #region

                        int[] r = new int[N];
                        for (int j = 0; j < Number; j++)
                        {
                            if (t == t0 || Number < 3)
                            {
                                r[j] = 1;
                            }
                            else
                            {
                                if (j < Number - 2)
                                {
                                    if ((Num_Velocity[j] <= Num_Velocity[j + 1] &&
Num_Velocity[j + 1] <= Num_Velocity[j + 2]) || Num_Velocity[j + 2] >=
Math.Max(Desired_speed[Num_Location[j + 2]], 19))
                                    {
                                        r[j] = 0;
                                    }
                                    else
                                    {
                                        r[j] = 1;
                                    }
                                }
                                else
                                {
                                    if (j == Number - 2)
                                    {
                                        if (Num_Velocity[j] < Num_Velocity[j + 1])
                                        {
                                            r[j] = 0;
                                        }
                                        else
                                        {
                                            r[j] = 1;
                                        }
                                    }
                                    else
```

```
                                {
                                    r[j] = 0;
                                }
                            }
                            if (Num_Location[j] > 4920 && tt > 29)
                                r[j] = 1;
                        }

                        #endregion

                        #region
                        int[] Delta = new int[N];
                        int[] t_f = new int[N];
                        int[] t_l = new int[N];
                        int[] aa = new int[N];
                        for (int j = 0; j < Number; j++)
                        {
                            t_l[j] = (int)Math.Floor(r[j] * (double)(Num_Velocity[j]) / D
+ (1 - r[j]) * Math.Min((double)(Num_Velocity[j]) / D, t_safe));
                            Delta[j] = Num_Velocity[j] * 2;
                        }
                        #endregion

                        #region

                        int[] C_n = new int[N];
                        int[] C_n1 = new int[N];
                        int[] left = new int[N];
                        int[] C_n_limit = new int[N];
                        int VC_n;
                        for (int j = 0; j < Number; j++)
                        {
                            if (j < Number - 1)
                            {
                                if (Num_Location[j] >= 4900)
                                {
                                    VC_n = Desired_speed[Num_Location[j]];
                                }
                                else
                                {
                                    VC_n = Desired_speed[Num_Location[j]];
                                }
                                for (C_n[j] = Math.Max(Num_Velocity[j] - 2, 0); C_n[j] <=
Math.Min(Num_Velocity[j] + 2, VC_n); C_n[j]++)
                                {
                                    t_f[j] = (int)Math.Ceiling(r[j] * (double)(C_n[j]) /
D + (1 - r[j]) * Math.Max(0, Math.Min((double)(C_n[j]) / D, t_safe) - 1));
                                    if (t_f[j] == 0)
                                    {
                                        aa[j] = 0;
                                    }
                                    else
                                    {
                                        aa[j] = 1;
                                    }
                                    if (C_n[j] % D == 0)
                                    {
```

```
                                    if ((t_f[j] + 1) * C_n[j] > (int)(Distance[j] +
Num_Velocity[j + 1] * t_l[j + 1] - 0.5 * (t_l[j + 1] + 1) * t_l[j + 1] * D + 0.5 *
(t_f[j] + 1) * t_f[j] * D))
                                    {
                                        C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                        break;
                                    }
                                    else
                                    {
                                        C_n1[j] = C_n[j];
                                        continue;
                                    }
                                }
                                else
                                {
                                    if ((t_f[j] + 1) * C_n[j] > (int)(Distance[j] +
Num_Velocity[j + 1] * t_l[j + 1] - 0.5 * (t_l[j + 1] + 1) * t_l[j + 1] * D + 0.5 *
(t_f[j] - 1) * t_f[j] * D + aa[j] * (2 * t_f[j] - 1)))
                                    {
                                        C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                        break;
                                    }
                                    else
                                    {
                                        C_n1[j] = C_n[j];
                                        continue;
                                    }
                                }
                            }
                        }
                        else
                        {
                            if (Num_Location[j] >= 4900)
                            {
                                VC_n = Desired_speed[Num_Location[j]];
                            }
                            else
                            {
                                VC_n = Desired_speed[Num_Location[j]];
                            }
                            if (signal_control == 1)
                            {
                                C_n1[j] = Num_Velocity[j] + 1;
                            }
                            else
                            {
                                if (signal_control == 2)
                                {
                                 if (Distance[j] == Vmax && Num_Location[j] >= 4995)
                                    {
                                        C_n1[j] = Num_Velocity[j] + 1;
                                    }
                                    else
                                    {
                                        for (C_n[j] = Math.Max(Num_Velocity[j] - 2,
0); C_n[j] <= Math.Min(Num_Velocity[j] + 2, VC_n); C_n[j]++)
                                        {
```

```
                                                t_f[j] = (int)Math.Ceiling(r[j] *
(double)(C_n[j]) / D + (1 - r[j]) * Math.Max(0, Math.Min((double)(C_n[j]) / D, t_safe) -
1));
                                                if (t_f[j] == 0)
                                                {
                                                    aa[j] = 0;
                                                }
                                                else
                                                {
                                                    aa[j] = 1;
                                                }
                                                if (C_n[j] % D == 0)
                                                {
                                                    if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] + 1) * t_f[j] * D))
                                                    {
                                                        C_n1[j] = Math.Max(C_n[j] - 1,
0);
                                                        break;
                                                    }
                                                    else
                                                    {
                                                        C_n1[j] = C_n[j];
                                                        continue;
                                                    }
                                                }
                                                else
                                                {
                                                    if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] - 1) * t_f[j] * D + aa[j] * (2 * t_f[j] - 1)))
                                                    {
                                                        C_n1[j] = Math.Max(C_n[j] - 1,
0);
                                                        break;
                                                    }
                                                    else
                                                    {
                                                        C_n1[j] = C_n[j];
                                                        continue;
                                                    }
                                                }
                                            }
                                        }
                                    }
                                    else
                                    {
                                        for (C_n[j] = Math.Max(Num_Velocity[j] - 2, 0);
C_n[j] <= Math.Min(Num_Velocity[j] + 2, VC_n); C_n[j]++)
                                        {
                                            t_f[j] = (int)Math.Ceiling(r[j] *
(double)(C_n[j]) / D + (1 - r[j]) * Math.Max(0, Math.Min((double)(C_n[j]) / D, t_safe) -
1));
                                            if (t_f[j] == 0)
                                            {
                                                aa[j] = 0;
                                            }
                                            else
                                            {
```

```
                                        aa[j] = 1;
                                    }
                                    if (C_n[j] % D == 0)
                                    {
                                        if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] + 1) * t_f[j] * D))
                                        {
                                            C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                            break;
                                        }
                                        else
                                        {
                                            C_n1[j] = C_n[j];
                                            continue;
                                        }
                                    }
                                    else
                                    {
                                        if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] - 1) * t_f[j] * D + aa[j] * (2 * t_f[j] - 1)))
                                        {
                                            C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                            break;
                                        }
                                        else
                                        {
                                            C_n1[j] = C_n[j];
                                            continue;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }


                #endregion

                #region
                int[] ad = new int[5000];
                if (tt > green && tt < green + 5)
                {
                    int cc = 0;
                    for (int j = Number - 1; j > 0; j--)
                    {
                        if (stgo[j] == 2)
                            cc++;
                        if (Num_Location[j] > 4920 && Num_Location[j] < 4995 &&
(j == Number - 1 || (stgo[j] == 2 && cc == 0)))
                        {
                            {
                             double gap = (double)Distance[j] / Num_Velocity[j];
                                int stbar = 4995 - Num_Location[j];
                                ad[j] = pp.AccDec(gap, stgo[j], stbar,
Num_Velocity[j]);
                            }
                        }
```

```
                        }
                    }

                    #endregion
                    #region
                    int[] Num_Velocity_Last = new int[N];
                    for (int j = 0; j < Number; j++)
                    {
                        switch (ad[j])
                        {
                            case 0:
                                Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Num_Velocity[j] + 1,
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                break;
                            case 1:
                                Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Num_Velocity[j] + 1,
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                break;
                            case 2:
                                Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Math.Max(Num_Velocity[j] - 1, 0),
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                break;
                            case 3:
                                Num_Velocity_Last[j] = Num_Velocity[j];
                                break;
                            case 4:
                                Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Math.Max(Num_Velocity[j] - 1, 0),
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                break;
                            case 5:
                                Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Math.Max(Num_Velocity[j] - 2, 0),
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                break;
                        }
                    }

                   #endregion
                    #region
                    for (int j = 0; j < Number; j++)
                    {
                        double sj = random.Next(0, 999);
                        sj /= 1000;
                        if (((signal_control == 1 || signal_control == 2) &&
Num_Location[j] >= 4995) || kkk[Number - 1] == 1)
                        {
                            continue;
                        }
                        if (sj < 0.16)
                        {
                            Num_Velocity_Last[j] = Math.Max(0,
Math.Max(Num_Velocity[j] - D, Num_Velocity_Last[j] - 1));
                        }
                    }
```

```
for (int j = 0; j < Number; j++)
{
    Num_Velocity[j] = Num_Velocity_Last[j];
}
#endregion
for (int j = Number - 1; j >= 0; j--)
{
    if (j < Number - 1)
    {
        if (Num_Velocity[j] - Num_Velocity[j + 1] > Distance[j])
        {
            Num_Velocity[j] = Num_Velocity[j + 1] + Distance[j];
        }
    }
    else
    {
        if (Num_Velocity[j] > Distance[j])
        {
            if (j == Number - 1 && Distance[j] < 0)
                Num_Velocity[j] = 0;
            else
                Num_Velocity[j] = Distance[j];
        }
    }
}

for (int j = 0; j < Number; j++)
{
    if (Num_Velocity[j] < 0)
    {
        error++;
    }
}
#region
k = 0;
kii = ki;
if (t >= TT1 - 2)
{
    for (int j = 0; j < Number; j++)
    {
        Velocity_2[j] = Velocity_1[j];
    }
}
for (int j = 0; j < Number; j++)
{
    Num_Location_1[j] = Num_Location[j] + Num_Velocity[j];
    Velocity_1[j] = Num_Velocity[j];

    if (j <= Number - 1 && j >= ki)
    {
        if (t > TT1 && Velocity_2[j - ki] - Num_Velocity[j] > 2)
        {
            Slam_Brake1[t - TT1 - 1]++;
            if (t_l22 < 5000)
            {
                t_l2[0, t_l22] = tt;
                t_l2[1, t_l22] = Num_Location[j];
                if (Num_Location[j] < 4900)
```

```
                                    error++;
                                t_l2[2, t_l22] = alpha;
                                t_l2[3, t_l22] = Desired_speed[Num_Location[j]];
                                t_l22++;
                            }
                        }
                    }
                    if (Num_Location_1[j] >= Road_Length)
                    {
                        k++;
                        if (t >= TT1)
                        {
                            Flux[(int)Math.Round(100 * (alpha - 0.01))]++;
                        }
                    }

                }
                if (t > TT1)
                {
                    Slam_Brake1[t - TT1 - 1] /= Number;
                }
                for (int j = 0; j < Number - k; j++)
                {
                    Desired_speed_1[Num_Location_1[j]] =
Desired_speed[Num_Location[j]];
                }
                for (int j = 0; j < Number; j++)
                {
                    Num_Location[j] = 0;
                }
                for (int j = 0; j < Number - k; j++)
                {
                    Num_Location[j] = Num_Location_1[j];
                }
                for (int j = 0; j < Road_Length; j++)
                {
                    Location[j] = 0;
                    Velocity[j] = 0;
                    Desired_speed[j] = 0;
                }
                for (int j = 0; j < Road_Length; j++)
                {
                    Desired_speed[j] = Desired_speed_1[j];
                }
                for (int j = 0; j < Number - k; j++)
                {
                    Location[Num_Location[j]] = 1;
                    Velocity[Num_Location[j]] = Num_Velocity[j];
                }
                #endregion

            #region
                #region
                #endregion

                if (Number > 5)
                {
                    for (int j = 0; j < Number - Math.Max(1, k); j++)
```

```
                            {
                                if (Num_Location[j] >= Num_Location[j + 1] &&
Num_Location[j + 1] != 0)
                                {
                                    error++;
                                }
                            }
                        }
                    }
                    #region

                    for (int j = 0; j < TT; j++)
                    {
                        DSZ1[(int)Math.Round(100 * (alpha - 0.01))] += DS1[j];
                        DSZ2[(int)Math.Round(100 * (alpha - 0.01))] += DS2[j];
                        Slam_Brake[(int)Math.Round(100 * (alpha - 0.01))] +=
Slam_Brake1[j];
                    }
                    DSZ1[(int)Math.Round(100 * (alpha - 0.01))] /= TT;
                    DSZ2[(int)Math.Round(100 * (alpha - 0.01))] /= TT;
                    Slam_Brake[(int)Math.Round(100 * (alpha - 0.01))] /= TT;

                    #endregion
                }
                for (int j = 0; j < 100; j++)
                {
                    DSZ11[j] += DSZ1[j];
                    DSZ22[j] += DSZ2[j];
                    Flux11[j] += Flux[j];
                    Slam_Brake11[j] += Slam_Brake[j];
                    if (DSZ11[j] == 0 || DSZ22[j] == 0)
                    {
                        error++;
                    }

                }

            }
            for (int j = 0; j < 100; j++)
            {
                DSZ11[j] /= CC;
                DSZ22[j] /= CC;
                Flux11[j] /= CC;
                Slam_Brake11[j] /= CC;
            }
            FileStream dsz1 = File.Create("E:\\DSZ1.txt");
            FileStream dsz2 = File.Create("E:\\DSZ2.txt");
            StreamWriter swdsz1 = new StreamWriter((System.IO.Stream)dsz1);
            StreamWriter swdsz2 = new StreamWriter((System.IO.Stream)dsz2);
            for (int j = 0; j < 100; j++)
            {
                swdsz1.Write("{0} ", DSZ11[j]);
                swdsz2.Write("{0} ", DSZ22[j]);
            }
            swdsz1.Close();
            swdsz2.Close();
            FileStream flow = File.Create("E:\\FLUX.txt");
            StreamWriter swflow = new StreamWriter((System.IO.Stream)flow);
```

```
        for (int j = 0; j < 100; j++)
        {
            swflow.Write("{0} ", Flux11[j]);
        }
        swflow.Close();

        FileStream brake = File.Create("E:\\Brake1.txt");
        StreamWriter swbrake = new StreamWriter((System.IO.Stream)brake);
        for (int j = 0; j < 100; j++)
        {
            swbrake.Write("{0} ", Slam_Brake11[j]);
        }
        swbrake.Close();
        FileStream st1 = File.Create("E:\\tl1.txt");
        StreamWriter stw1 = new StreamWriter((System.IO.Stream)st1);
        FileStream st2 = File.Create("E:\\tl2.txt");
        StreamWriter stw2 = new StreamWriter((System.IO.Stream)st2);
        for (int jj = 0; jj < 4; jj++)
        {
            for (int j = 0; j < 5000; j++)
            {
                stw1.Write("{0} ", t_l1[jj, j]);
                stw2.Write("{0} ", t_l2[jj, j]);
            }
            stw1.Write("\n");
            stw2.Write("\n");
        }
        stw1.Close();
        stw2.Close();
        FileStream stgo_Error = File.Create("E:\\stgo_error.txt");
        StreamWriter sger = new StreamWriter((System.IO.Stream)stgo_Error);
        for (int j = 0; j < stgo_para; j++)
        {
            sger.Write("{0} ", stgo_error[j]);
        }
        sger.Close();
                    #endregion

        Console.WriteLine("OVER");
        Console.ReadLine();
    }

    public int StopGo(int di, int sp, bool lf)
    {

        #region
        Random aa = new Random();
        if (di > 4941)
        {
            double cc = aa.Next(0, 999) / 1000;
            if (cc < 0.191)
                return 2;
            else
                return 1;
        }
        else
            if (di < 4924)
            {
```

```
            double cc = aa.Next(0, 999) / 1000;
            if (cc < 0.922)
                return 2;
            else
                return 1;
        }
        else
            if (sp < 15)
            {
                double cc = aa.Next(0, 999) / 1000;
                if (cc < 0.604)
                    return 2;
                else
                    return 1;
            }
            else
                if (lf)// leading 为ture following 为flase
                {
                    double cc = aa.Next(0, 999) / 1000;
                    if (cc < 0.488)
                        return 2;
                    else
                        return 1;
                }
                else
                {
                    double cc = aa.Next(0, 999) / 1000;
                    if (cc < 59.6)
                        return 2;
                    else
                        return 1;
                }
    #endregion
}

public int AccDec(double gap, int stopgo, int Dtostopbar, int speed)
{
    Random pa = new Random();
    if (stopgo == 1)
    {
        if (gap <= 3)
        {
            double aa = pa.Next(0, 1000);
            aa /= 1000;
            if (aa < 0.7686)
                return 2;
            else
                return 3;
        }
        else if (gap >= 3.6)
        {
            double aa = pa.Next(0, 1000);
            aa /= 1000;
            if (aa < 0.07)
                return 1;
            else
                return 3;
```

```
                }
                else
                    return 3;
            }
            else
            {
                int d1, d2;
                d1 = speed * (speed - 1) / 2;
                if (speed % 2 == 0)
                    d2 = speed * (speed - 2) / 4;
                else
                    d2 = (speed - 1) * (speed - 1) / 4;
                if (Dtostopbar > d1)
                {
                    if (Dtostopbar - speed > d1)
                        return 0;
                    else if (Dtostopbar - speed - 1 > (speed - 1) * (speed - 2) / 2)
                    {
                        return 4;
                    }
                    else
                    {
                        return 5;
                    }
                }
                else if (Dtostopbar > d2)
                    return 5;
                else
                    return 5;
            }
        }

        public double ST_GO_LogitRegression(bool lead, int spe, int dis)
        {
            int x1 = 0, x2 = 0, x3 = 0, x4 = 0, x5 = 0, x6 = 0;
            if (lead)
                x1 = 1;

            if (spe > 14 && spe < 16)
                x2 = 1;
            else
                if (spe < 20)
                    x3 = 1;
            if (dis > 57 && dis < 79)
                x4 = 1;
            else
                if (dis >= 79 && dis < 88)
                    x5 = 1;
                else
                    if (dis >= 88 && dis < 98)
                        x6 = 1;

            double y = Math.Exp(-0.1945 + 0.9350 * x1 + 1.4994 * x2 + 3.2820 * x3 -
2.4108 * x4 - 4.5557 * x5 - 5.2498 * x6) /
                (1 + Math.Exp(-0.1945 + 0.9350 * x1 + 1.4994 * x2 + 3.2820 * x3 - 2.4108
* x4 - 4.5557 * x5 - 5.2498 * x6));
            return y;
        }
```

```
        public double AverageRandom(double min, double max)
        {
            double aa = random.Next(0, 1001);
            aa = aa / 1000;
            return min + aa * (max - min);
        }

        public double Normal(double x, double mu, double sigma)
        {
            return 1.0 / Math.Sqrt(2 * Math.PI * sigma) * Math.Exp(-1 * (x - mu) * (x -
mu) / (2 * sigma * sigma));
        }

        public int NormalRandom()
        {
            double x;
            double y;
            double dScope;
            do
            {
                x = AverageRandom(34, 66);
                dScope = Normal(x, 50, 5);
                y = AverageRandom(0, 0.14);
            }
            while (y >= dScope);
            x = (int)Math.Round(x * 0.44704 / 1.5, 0);
            return (int)x;
        }
    }
}
```

APPENDIX B C# CODE FOR THE INTERSECTION FOR THE INTERSECTION WITH THE FLASHING GREEN PHASES

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
namespace Amber_Flash
{
    class Program
    {
        Random random = new Random();
        static void Main(string[] args)
        {
            Random random = new Random();
            int Road_Length = 5000;
            int Vmax = 20;
            int Vfast = 20;
            double[] P = { 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 };
            int t0 = 0;
            int TT1 = 10000;
            int TT = 5000;
            double alpha;
            int L_veh = 5;
            int green = 25;
            int C = 60;
            int red = 25;
            int ki;
            int kii = 0;
            int error = 0;
            int D = 2;
            int t_safe = 3;
            int Vlimit = 15;
            double[,] t_l1 = new double[4, 5000];
            double[,] t_l2 = new double[4, 5000];
            int t_l11 = 0;
            int t_l22 = 0;
            double[] DSZ11 = new double[100];
            double[] DSZ22 = new double[100];
            int[] Flux11 = new int[100];
            double[] Slam_Brake11 = new double[100];
            double[] stgo_error = new double[10000];
            int stgo_para = 0;
            int CC = 1;
            int k = 0;
            for (int mm = 0; mm < CC; mm++)
            {
                double[] DSZ1 = new double[100];
                double[] DSZ2 = new double[100];
                int[] Flux = new int[100];
                double[] Slam_Brake = new double[100];
                int[,] Time_Location = new int[300, Road_Length];
                int[,] ArrTL = new int[240, 2000];

                int M = 0;
                int Dis = 0;
                int para_signal = 0;
                Program pm = new Program();
                for (alpha = 0.01; alpha < 0.401; alpha += 0.01)
                {
```

```
#region
int[] Location = new int[Road_Length];
int[] Velocity = new int[Road_Length];
double[] DS1 = new double[TT];
double[] DS2 = new double[TT];
double[] Slam_Brake1 = new double[TT];
int[] Desired_speed = new int[5000];
int[] Desired_speed_1 = new int[5000];
int[] Velocity_1 = new int[Road_Length];
int[] stgo_1 = new int[Road_Length];
int[] Velocity_2 = new int[Road_Length];
int[] stgo = new int[5000];
int signal_control = 1;
int tt = 0;
#endregion
for (int t = t0; t < (TT1 + TT); t++)
{
    if (tt == green + 5)
    {
        for (int ii = 0; ii < Road_Length; ii++)
        {
            stgo[ii] = 0;
        }
    }
    #region
    if (t > t0)
    {
        if (tt >= C)
        {
            tt = tt - C;
        }
        tt = tt + 1;
        if (tt <= green)
        {
            signal_control = 1;
        }
        else
        {
            if (tt <= green + 4)
            {
                signal_control = 2;
            }
            else
            {
                signal_control = 3;
            }
        }

    }
    #endregion

    #region
    if (t % 60 == 0)
    {
        double cs = random.Next(3, 6);
        int born_number = (int)(Math.Floor(cs));
        int dspeed = pm.NormalRandom();
        ki = born_number;
```

```
        for (int ii = 0; ii < born_number; ii++)
        {
            Location[ii * (dspeed + L_veh)] = 1;
            Velocity[ii * (dspeed + L_veh)] = dspeed;
            Desired_speed[ii * (dspeed + L_veh)] = dspeed;
        }
    }
    else
        ki = 0;
    #endregion

    #region
    int N = 0;
    for (int j = 0; j < Road_Length; j++)
    {
        if (Location[j] == 1)
        {
            N++;
        }
    }
    if (N == 0)
    {
        continue;
    }
    if (t == 3000)
    {
        t = 3000;

    }
    int[] Location_1 = new int[N];
    int[] Num_Location_1 = new int[N];
#endregion
    #region
    if (tt > green && tt < green + 5)
    {
        for (int j = 0; j < Road_Length; j++)
        {
            stgo_1[j] = stgo[j];
        }
        for (int j = 0; j < Road_Length; j++)
        {
            stgo[j] = 0;
        }
        for (int j = 0; j < Road_Length - 1; j++)
        {
            stgo[j + ki] = stgo_1[j];
        }
    }
    #endregion
    #region

    int[] Distance = new int[N];
    int[] Num_Location = new int[N];
    int[] Num_Velocity = new int[N];
    int[] Virtual_Velocity = new int[N];
    int[] kkk = new int[N];
    int Number = 1;
    int i = 0;
```

```
while (i < Road_Length)
{
    int[] a = new int[2];
    int c = 0;
    while (i < Road_Length && c < 2)
    {
        if (Location[i] == 1)
        {
            a[c] = i;
            c++;
        }
        i++;
    }
    if (c == 2)
    {
        i--;
        c = 0;
        Distance[Number - 1] = a[1] - a[0] - L_veh;
        Num_Location[Number - 1] = a[0];
        Num_Velocity[Number - 1] = Velocity[a[0]];
        Number++;
    }
    else
    {
        if (a[0] >= 4995)
        {
            Distance[Number - 1] = Vmax;
            switch (signal_control)
            {
                case 1:
                    Distance[Number - 1] = Vmax;
                    break;
                case 2:
                    if (stgo[Number - 1] == 1)
                        Distance[Number - 1] = Vmax;
                    else
                        Distance[Number - 1] = 0;
                    break;
                case 3:
                    if (tt <= green + 5)
                        Distance[Number - 1] = Vmax;
                    else
                        Distance[Number - 1] = 0;
                    break;
            }
        }
        else
        {
            if (signal_control == 1)
            {
                Distance[Number - 1] = Vmax;
            }
            else
            {
                if (signal_control == 2)
                {
                    if (stgo[Number - 1] == 1)
                        Distance[Number - 1] = Vmax;
```

```
                    else
                Distance[Number - 1] = Road_Length - a[0] - L_veh;
                    }
                    else
                    {
                     Distance[Number - 1] = Road_Length - a[0] - L_veh;
                    }

                    }
                }
                Num_Location[Number - 1] = a[0];
                Num_Velocity[Number - 1] = Velocity[a[0]];
                break;
            }
        }
        for (int j = 0; j < Number; j++)
        {
            if (Distance[j] < 0)
            {
                error++;
            }
        }
        #endregion

        for (int j = 0; j < Number; j++)
        {
            if (Distance[j] < 0)
            {
                error++;
            }
        }

        for (int j = 0; j < Number - 1; j++)
        {
            if (Num_Velocity[j] - Num_Velocity[j + 1] > Distance[j])
            {
                error++;
            }
        }



        #region
        if (t > TT1)
        {
            for (int j = ki + kii; j < Number - 1; j++)
            {
                if (Num_Velocity[j] < Vmax && Distance[j] == 0)
                {
                    if (Velocity_2[j + 1 - ki - kii] >= 0)
                    {
                        if (Velocity_2[j - ki - kii] - Num_Velocity[j] >
2)

                        {

                            if (Num_Velocity[j + 1] == 0)
                            {
                                DS1[t - TT1 - 1] += 1;
```

```
                                        }
                                        else
                                        {
                                            DS2[t - TT1 - 1] += 1;
                                        }
                                        if (t_l11 < 5000)
                                        {
                                            t_l1[0, t_l11] = tt;
                                            t_l1[1, t_l11] = Num_Location[j];
                                            t_l1[2, t_l11] = alpha;
                                            t_l1[3, t_l11] =
Desired_speed[Num_Location[j]];
                                            t_l11++;
                                        }
                                    }
                                }
                            }
                            DS1[t - TT1 - 1] /= (Number - 1);
                            DS2[t - TT1 - 1] /= (Number - 1);
                        }
                        #endregion

                        Program pp = new Program();

                        #region

                        int[] Dec = new int[5000];
                                        if (tt == green + 2)
                        {
                            for (int j = 0; j < Number; j++)
                            {
                                int Num_pdcar = 0;
                                int Num_stgo_error = 0;

                                if (Num_Location[j] > 4900 && Num_Location[j] < 4995)
                                {
                                    Num_pdcar++;

                                    double sj = random.Next(0, 999);
                                    sj /= 1000;
                                    bool lead = false;
                                    if (j == Number - 1)
                                        lead = true;
                                    double jj = pp.ST_GO_LogitRegression(lead,
Num_Velocity[j], 4995 - Num_Location[j]);
                                    if (jj > 0.48)
                                        stgo[j] = 1;
                                    else
                                        stgo[j] = 2;

                                    if (stgo[j] == 1 && ((5000 - Num_Location[j]) > 4 *
Num_Velocity[j]))
                                        Num_stgo_error++;

                                }
                                else
                                {
```

```
                                stgo[j] = 2;
                            }


                            if (Num_pdcar > 0)
                            {
                                stgo_error[stgo_para] = (double)Num_stgo_error /
Num_pdcar;
                                stgo_para++;
                            }
                        }
                    }
                    #endregion
                    #region

                    int[] r = new int[N];/
                    for (int j = 0; j < Number; j++)
                    {
                        if (t == t0 || Number < 3)
                        {
                            r[j] = 1;
                        }
                        else
                        {
                            if (j < Number - 2)
                            {
                                if ((Num_Velocity[j] <= Num_Velocity[j + 1] &&
Num_Velocity[j + 1] <= Num_Velocity[j + 2]) || Num_Velocity[j + 2] >=
Math.Max(Desired_speed[Num_Location[j + 2]], 19))
                                {
                                    r[j] = 0;
                                }
                                else
                                {
                                    r[j] = 1;
                                }
                            }
                            else
                            {
                                if (j == Number - 2)
                                {
                                    if (Num_Velocity[j] < Num_Velocity[j + 1])
                                    {
                                        r[j] = 0;
                                    }
                                    else
                                    {
                                        r[j] = 1;
                                    }
                                }
                                else
                                {
                                    r[j] = 0;
                                                                                }
                                }
                            }
                            if (Num_Location[j] > 4920 && tt > 29)
                                r[j] = 1;
```

```
                        }
                        #endregion
                        #region
                        int[] Delta = new int[N];
                        int[] t_f = new int[N];
                        int[] t_l = new int[N];
                        int[] aa = new int[N];
                        for (int j = 0; j < Number; j++)
                        {
        t_l[j] = (int)Math.Floor(r[j] * (double)(Num_Velocity[j]) / D + (1 - r[j]) *
Math.Min((double)(Num_Velocity[j]) / D, t_safe));

                            Delta[j] = Num_Velocity[j] * 2;
                        }
                        #endregion

                        #region
                        int[] C_n = new int[N];
                        int[] C_n1 = new int[N];
                        int[] left = new int[N];
                        int[] C_n_limit = new int[N];
                        int VC_n;
                        for (int j = 0; j < Number; j++)
                        {
                            if (j < Number - 1)
                            {
                                if (Num_Location[j] >= 4900)
                                {
                                    VC_n = Desired_speed[Num_Location[j]];
                                }
                                else
                                {
                                    VC_n = Desired_speed[Num_Location[j]];
                                }
                                for (C_n[j] = Math.Max(Num_Velocity[j] - 2, 0); C_n[j] <=
Math.Min(Num_Velocity[j] + 2, VC_n); C_n[j]++)
                                {
                                    t_f[j] = (int)Math.Ceiling(r[j] * (double)(C_n[j]) /
D + (1 - r[j]) * Math.Max(0, Math.Min((double)(C_n[j]) / D, t_safe) - 1));
                                    if (t_f[j] == 0)
                                    {
                                        aa[j] = 0;
                                    }
                                    else
                                    {
                                        aa[j] = 1;
                                    }
                                    if (C_n[j] % D == 0)
                                    {
                                        if ((t_f[j] + 1) * C_n[j] > (int)(Distance[j] +
Num_Velocity[j + 1] * t_l[j + 1] - 0.5 * (t_l[j + 1] + 1) * t_l[j + 1] * D + 0.5 *
(t_f[j] + 1) * t_f[j] * D))
                                        {
                                            C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                            break;
                                        }
                                        else
                                        {
```

```
                                        C_n1[j] = C_n[j];
                                        continue;
                                    }
                                }
                                else
                                {
                                    if ((t_f[j] + 1) * C_n[j] > (int)(Distance[j] +
Num_Velocity[j + 1] * t_l[j + 1] - 0.5 * (t_l[j + 1] + 1) * t_l[j + 1] * D + 0.5 *
(t_f[j] - 1) * t_f[j] * D + aa[j] * (2 * t_f[j] - 1)))
                                    {
                                        C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                        break;
                                    }
                                    else
                                    {
                                        C_n1[j] = C_n[j];
                                        continue;
                                    }
                                }
                            }
                        }
                        else
                        {
                            if (Num_Location[j] >= 4900)
                            {
                                VC_n = Desired_speed[Num_Location[j]];
                            }
                            else
                            {
                                VC_n = Desired_speed[Num_Location[j]];
                            }
                            if (signal_control == 1)
                            {
                                C_n1[j] = Num_Velocity[j] + 1;
                            }
                            else
                            {
                                if (signal_control == 2)
                                {
                                 if (Distance[j] == Vmax && Num_Location[j] >= 4995)
                                    {
                                        C_n1[j] = Num_Velocity[j] + 1;
                                    }
                                    else
                                    {
                                      for (C_n[j] = Math.Max(Num_Velocity[j] - 2, 0);
C_n[j] <= Math.Min(Num_Velocity[j] + 2, VC_n); C_n[j]++)
                                        {
                                            t_f[j] = (int)Math.Ceiling(r[j] *
(double)(C_n[j]) / D + (1 - r[j]) * Math.Max(0, Math.Min((double)(C_n[j]) / D, t_safe) -
1));
                                            if (t_f[j] == 0)
                                            {
                                                aa[j] = 0;
                                            }
                                            else
                                            {
                                                aa[j] = 1;
```

```
                                                }
                                                if (C_n[j] % D == 0)
                                                {
                                                    if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] + 1) * t_f[j] * D))
                                                    {
                                                        C_n1[j] = Math.Max(C_n[j] - 1,
0);
                                                        break;
                                                    }
                                                    else
                                                    {
                                                        C_n1[j] = C_n[j];
                                                        continue;
                                                    }
                                                }
                                                else
                                                {
                                                    if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] - 1) * t_f[j] * D + aa[j] * (2 * t_f[j] - 1)))
                                                    {
                                                        C_n1[j] = Math.Max(C_n[j] - 1,
0);
                                                        break;
                                                    }
                                                    else
                                                    {
                                                        C_n1[j] = C_n[j];
                                                        continue;
                                                    }
                                                }
                                            }
                                        }
                                    }
                                    else
                                    {
                                for (C_n[j] = Math.Max(Num_Velocity[j] - 2, 0);
C_n[j] <= Math.Min(Num_Velocity[j] + 2, VC_n); C_n[j]++)
                                        {
                                            t_f[j] = (int)Math.Ceiling(r[j] *
(double)(C_n[j]) / D + (1 - r[j]) * Math.Max(0, Math.Min((double)(C_n[j]) / D, t_safe) -
1));
                                            if (t_f[j] == 0)
                                            {
                                                aa[j] = 0;
                                            }
                                            else
                                            {
                                                aa[j] = 1;
                                            }
                                            if (C_n[j] % D == 0)
                                            {
        if ((t_f[j] + 1) * C_n[j] > (int)(Distance[j] + 0.5 * (t_f[j] + 1) * t_f[j] * D))
                                                {
                                                    C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                                    break;
                                                }
                                                else
```

```
                                        {
                                            C_n1[j] = C_n[j];
                                            continue;
                                        }
                                    }
                                    else
                                    {
                                        if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] - 1) * t_f[j] * D + aa[j] * (2 * t_f[j] - 1)))
                                        {
                                            C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                            break;
                                        }
                                        else
                                        {
                                            C_n1[j] = C_n[j];
                                            continue;
                                        }
                                    }
                                }
                            }
                        }
                    }

                    #endregion
                   #region
                    int[] ad = new int[5000];
                    if (tt > green && tt < green + 5)
                    {
                        int cc = 0;
                        for (int j = Number - 1; j > 0; j--)
                        {
                            if (stgo[j] == 2)
                                cc++;
                            if (Num_Location[j] > 4920 && Num_Location[j] < 4995 &&
(j == Number - 1 || (stgo[j] == 2 && cc == 0)))
                            {
                                {
                                 double gap = (double)Distance[j] / Num_Velocity[j];
                                        int stbar = 4995 - Num_Location[j];
                                        ad[j] = pp.AccDec(gap, stgo[j], stbar,
Num_Velocity[j]);
                                }
                            }
                        }
                    }

                    #endregion

                    #region
                    int[] Num_Velocity_Last = new int[N];
                    for (int j = 0; j < Number; j++)
                    {
                        switch (ad[j])
                        {
                            case 0:
```

```
                                    Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Num_Velocity[j] + 1,
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                    break;
                                case 1:
                                    Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Num_Velocity[j] + 1,
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                    break;
                                case 2:
                                    Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Math.Max(Num_Velocity[j] - 1, 0),
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                    break;
                                case 3:
                                    Num_Velocity_Last[j] = Num_Velocity[j];
                                    break;
                                case 4:
                                    Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Math.Max(Num_Velocity[j] - 1, 0),
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                    break;
                                case 5:
                                    Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Math.Max(Num_Velocity[j] - 2, 0),
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                    break;
                            }
                        }


                        #endregion
                        #region

                        for (int j = 0; j < Number; j++)
                        {
                            Num_Velocity_Last[j] = Num_Velocity_Last[j] -
Dec[Num_Location[j]];
                        }
                        #endregion
                        #region
                        for (int j = 0; j < Number; j++)
                        {
                            double sj = random.Next(0, 999);
                            sj /= 1000;
                            if (((signal_control == 1 || signal_control == 2) &&
Num_Location[j] >= 4995) || kkk[Number - 1] == 1)
                            {
                                continue;
                            }
                            if (sj < 0.16)
                            {
                                Num_Velocity_Last[j] = Math.Max(0,
Math.Max(Num_Velocity[j] - D, Num_Velocity_Last[j] - 1));
                            }
                            if (Num_Velocity_Last[j] > Num_Velocity[j] + 1)
                            {
                                Num_Velocity[j] = Num_Velocity[j] + 1;
```

```
    }
}
for (int j = 0; j < Number; j++)
{
    Num_Velocity[j] = Num_Velocity_Last[j];
}
#endregion

for (int j = Number - 1; j >= 0; j--)
{
    if (j < Number - 1)
    {
        if (Num_Velocity[j] - Num_Velocity[j + 1] > Distance[j])
        {
            Num_Velocity[j] = Num_Velocity[j + 1] + Distance[j];
        }
    }
    else
    {
        if (Num_Velocity[j] > Distance[j])
        {
            if (j == Number - 1 && Distance[j] < 0)
                Num_Velocity[j] = 0;
            else
                Num_Velocity[j] = Distance[j];
        }
    }
}

for (int j = 0; j < Number; j++)
{
    if (Num_Velocity[j] < 0)
    {
        error++;
    }
}
#region
k = 0;
kii = ki;
if (t >= TT1 - 2)
{
    for (int j = 0; j < Number; j++)
    {
        Velocity_2[j] = Velocity_1[j];
    }
}
for (int j = 0; j < Number; j++)
{
    Num_Location_1[j] = Num_Location[j] + Num_Velocity[j];
    Velocity_1[j] = Num_Velocity[j];

    if (j <= Number - 1 && j >= ki)
    {
        if (t > TT1 && Velocity_2[j - ki] - Num_Velocity[j] > 2)
        {
            Slam_Brake1[t - TT1 - 1]++;
            if (t_l22 < 5000)
            {
```

```
                        t_l2[0, t_l22] = tt;
                        t_l2[1, t_l22] = Num_Location[j];
                        if (Num_Location[j] < 4900)
                            error++;
                        t_l2[2, t_l22] = alpha;
                        t_l2[3, t_l11] = Desired_speed[Num_Location[j]];
                        t_l22++;
                    }
                }
            }
            if (Num_Location_1[j] >= Road_Length)
            {
                k++;
                if (t >= TT1)
                {
                    Flux[(int)Math.Round(100 * (alpha - 0.01))]++;
                }
            }

        }
        if (t > TT1)
        {
            Slam_Brake1[t - TT1 - 1] /= Number;
        }
        for (int j = 0; j < Number - k; j++)
        {
    Desired_speed_1[Num_Location_1[j]] = Desired_speed[Num_Location[j]];
        }
        for (int j = 0; j < Number; j++)
        {
            Num_Location[j] = 0;
        }
        for (int j = 0; j < Number - k; j++)
        {
            Num_Location[j] = Num_Location_1[j];
        }
        for (int j = 0; j < Road_Length; j++)
        {
            Location[j] = 0;
            Velocity[j] = 0;
            Desired_speed[j] = 0;
        }
        for (int j = 0; j < Road_Length; j++)
        {
            Desired_speed[j] = Desired_speed_1[j];
        }
        for (int j = 0; j < Number - k; j++)
        {
            Location[Num_Location[j]] = 1;
            Velocity[Num_Location[j]] = Num_Velocity[j];
        }
        #endregion
        if (Number > 5)
        {
            for (int j = 0; j < Number - Math.Max(1, k); j++)
            {
    if (Num_Location[j] >= Num_Location[j + 1] && Num_Location[j + 1] != 0)
            {
```

```
                        error++;
                    }
                }
            }
        }
        #region
        for (int j = 0; j < TT; j++)
        {
            DSZ1[(int)Math.Round(100 * (alpha - 0.01))] += DS1[j];
            DSZ2[(int)Math.Round(100 * (alpha - 0.01))] += DS2[j];
        Slam_Brake[(int)Math.Round(100 * (alpha - 0.01))] += Slam_Brake1[j];
        }
        DSZ1[(int)Math.Round(100 * (alpha - 0.01))] /= TT;
        DSZ2[(int)Math.Round(100 * (alpha - 0.01))] /= TT;
        Slam_Brake[(int)Math.Round(100 * (alpha - 0.01))] /= TT;
        #endregion
    }
    for (int j = 0; j < 100; j++)
    {
        DSZ11[j] += DSZ1[j];
        DSZ22[j] += DSZ2[j];
        Flux11[j] += Flux[j];
        Slam_Brake11[j] += Slam_Brake[j];
        if (DSZ11[j] == 0 || DSZ22[j] == 0)
        {
            error++;
        }
    }
}
for (int j = 0; j < 100; j++)
{
    DSZ11[j] /= CC;
    DSZ22[j] /= CC;
    Flux11[j] /= CC;
    Slam_Brake11[j] /= CC;
}
// dangerous situation
FileStream dsz1 = File.Create("E:\\DSZ1.txt");
FileStream dsz2 = File.Create("E:\\DSZ2.txt");
StreamWriter swdsz1 = new StreamWriter((System.IO.Stream)dsz1);
StreamWriter swdsz2 = new StreamWriter((System.IO.Stream)dsz2);
for (int j = 0; j < 100; j++)
{
    swdsz1.Write("{0} ", DSZ11[j]);
    swdsz2.Write("{0} ", DSZ22[j]);
}
swdsz1.Close();
swdsz2.Close();
// flux
FileStream flow = File.Create("E:\\FLUX.txt");
StreamWriter swflow = new StreamWriter((System.IO.Stream)flow);
for (int j = 0; j < 100; j++)
{
    swflow.Write("{0} ", Flux11[j]);
}
swflow.Close();
FileStream brake = File.Create("E:\\Brake1.txt");
StreamWriter swbrake = new StreamWriter((System.IO.Stream)brake);
```

```
    for (int j = 0; j < 100; j++)
    {
        swbrake.Write("{0} ", Slam_Brake11[j]);
    }
    swbrake.Close();
    FileStream st1 = File.Create("E:\\tl1.txt");
    StreamWriter stw1 = new StreamWriter((System.IO.Stream)st1);
    FileStream st2 = File.Create("E:\\tl2.txt");
    StreamWriter stw2 = new StreamWriter((System.IO.Stream)st2);
    for (int jj = 0; jj < 4; jj++)
    {
        for (int j = 0; j < 5000; j++)
        {
            stw1.Write("{0} ", t_l1[jj, j]);
            stw2.Write("{0} ", t_l2[jj, j]);
        }
        stw1.Write("\n");
        stw2.Write("\n");
    }
    stw1.Close();
    stw2.Close();
    FileStream stgo_Error = File.Create("E:\\stgo_error.txt");
    StreamWriter sger = new StreamWriter((System.IO.Stream)stgo_Error);
    for (int j = 0; j < stgo_para; j++)
    {
        sger.Write("{0} ", stgo_error[j]);
    }
    sger.Close();
                #endregion
    Console.WriteLine("OVER");
    Console.ReadLine();
}
public int StopGo(int di, int sp, bool lf)
{
    #region
    Random aa = new Random();
    if (di > 4941)
    {
        double cc = aa.Next(0, 999) / 1000;
        if (cc < 0.191)
            return 2;
        else
            return 1;
    }
    else
        if (di < 4924)
        {
            double cc = aa.Next(0, 999) / 1000;
            if (cc < 0.922)
                return 2;
            else
                return 1;
        }
        else
            if (sp < 15)
            {
                double cc = aa.Next(0, 999) / 1000;
                if (cc < 0.604)
```

```
                    return 2;
                else
                    return 1;
            }
            else
                if (lf)
                {
                    double cc = aa.Next(0, 999) / 1000;
                    if (cc < 0.488)
                        return 2;
                    else
                        return 1;
                }
                else
                {
                    double cc = aa.Next(0, 999) / 1000;
                    if (cc < 59.6)
                        return 2;
                    else
                        return 1;
                }
    #endregion
}

public int AccDec(double gap, int stopgo, int Dtostopbar, int speed)
{
    Random pa = new Random();
    if (stopgo == 1)
    {
        if (gap <= 3)
        {
            double aa = pa.Next(0, 1000);
            aa /= 1000;
            if (aa < 0.7686)
                return 2;
            else
                return 3;
        }
        else if (gap >= 3.6)
        {
            double aa = pa.Next(0, 1000);
            aa /= 1000;
            if (aa < 0.07)
                return 1;
            else
                return 3;
        }
        else
            return 3;
    }
    else
    {
        int d1, d2;
        d1 = speed * (speed - 1) / 2;
        if (speed % 2 == 0)
            d2 = speed * (speed - 2) / 4;
        else
            d2 = (speed - 1) * (speed - 1) / 4;
```

```
                if (Dtostopbar > d1)
                {
                    if (Dtostopbar - speed > d1)
                        return 0;
                    else if (Dtostopbar - speed - 1 > (speed - 1) * (speed - 2) / 2)
                    {
                        return 4;
                    }
                    else
                    {
                        return 5;
                    }
                }
                else if (Dtostopbar > d2)
                    return 5;
                else
                    return 5;
            }
        }

        public double ST_GO_LogitRegression(bool lead, int spe, int dis)
        {
            int x1 = 0, x2 = 0, x3 = 0, x4 = 0, x5 = 0, x6 = 0;
            if (lead)
                x1 = 1;

            if (spe > 14 && spe < 16)
                x2 = 1;
            else
                if (spe < 20)
                    x3 = 1;
            if (dis > 57 && dis < 79)
                x4 = 1;
            else
                if (dis >= 79 && dis < 88)
                    x5 = 1;
                else
                    if (dis >= 88 && dis < 98)
                        x6 = 1;

            double y = Math.Exp(-0.1945 + 0.9350 * x1 + 1.4994 * x2 + 3.2820 * x3 -
2.4108 * x4 - 4.5557 * x5 - 5.2498 * x6) /
                (1 + Math.Exp(-0.1945 + 0.9350 * x1 + 1.4994 * x2 + 3.2820 * x3 - 2.4108
* x4 - 4.5557 * x5 - 5.2498 * x6));
            return y;
        }

        public double AverageRandom(double min, double max)
        {
            double aa = random.Next(0, 1001);
            aa = aa / 1000;
            return min + aa * (max - min);
        }

        public double Normal(double x, double mu, double sigma)
        {
            return 1.0 / Math.Sqrt(2 * Math.PI * sigma) * Math.Exp(-1 * (x - mu) * (x -
mu) / (2 * sigma * sigma));
```

```
        }

        public int NormalRandom()
        {
            double x;
            double y;
            double dScope;
            do
            {
                x = AverageRandom(34, 66);
                dScope = Normal(x, 50, 5);
                y = AverageRandom(0, 0.14);
            }
            while (y >= dScope);
            x = (int)Math.Round(x * 0.44704 / 1.5, 0);
            return (int)x;
        }

        public int Acc_NormalRandom()
        {
            double x;
            double y;
            double dScope;
            do
            {
                x = AverageRandom(-0.5, 2.5);
                dScope = Normal(x, 1, 0.7);
                y = AverageRandom(0, 0.18);
            }
            while (y >= dScope);
            return (int)(Math.Round(x, 0));
        }
    }
}
```

APPENDIX C C# CODE FOR THE PAVEMENT MARKING SCENARIO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
namespace Pavement_marking
{
    class Program
    {
        Random random = new Random();
        static void Main(string[] args)
        {
            Random random = new Random();
            int Road_Length = 5000;
            int Vmax = 20;
            int Vfast = 20;
            double[] P = { 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 };;
            int t0 = 0;
            int TT1 = 10000;
            int TT = 5000;
            double alpha;
            int L_veh = 5;
            int green = 25;
            int C = 60;
            int red = 25;
            int ki;
            int kii = 0;
            int error = 0;
            int D = 2;
            int t_safe = 3;
            int Vlimit = 15;
            double[,] t_l1 = new double[4, 5000];
            double[,] t_l2 = new double[4, 5000];
            int t_l11 = 0;
            int t_l22 = 0;
            double[] DSZ11 = new double[100];
            double[] DSZ22 = new double[100];
            int[] Flux11 = new int[100];
            double[] Slam_Brake11 = new double[100];
            double[] stgo_error = new double [10000];
            int stgo_para = 0;
            int CC = 1;
            int k = 0;
            for (int mm = 0; mm < CC; mm++)
            {
                double[] DSZ1 = new double[100];
                double[] DSZ2 = new double[100];
                int[] Flux = new int[100];
                double[] Slam_Brake = new double[100];
                int[,] Time_Location = new int[300, Road_Length];
                int[,] ArrTL = new int[240, 2000];
                int M = 0;
                int Dis = 0;
                int para_signal = 0;
                Program pm = new Program();
                for (alpha = 0.01; alpha < 0.401; alpha += 0.01)
                {
                    #region
```

```
int[] Location = new int[Road_Length];
int[] Velocity = new int[Road_Length];
double[] DS1 = new double[TT];
double[] DS2 = new double[TT];
double[] Slam_Brake1 = new double[TT];
int[] Desired_speed = new int[5000];
int[] Desired_speed_1 = new int[5000];
int[] Velocity_1 = new int[Road_Length];
int[] stgo_1 = new int[Road_Length];
int[] Velocity_2 = new int[Road_Length];
int[] stgo = new int[5000];
int signal_control = 1;
int tt = 0;

#endregion
//
for (int t = t0; t < (TT1 + TT); t++)
{
    if (tt == green + 5)
    {
        for (int ii = 0; ii < Road_Length; ii++)
        {
            stgo[ii] = 0;
        }
    }

    #region
    if (t > t0)
    {
        if (tt >= C)
        {
            tt = tt - C;
        }
        tt = tt + 1;
        if (tt <= green)
        {
            signal_control = 1;
        }
        else
        {
            if (tt <= green + 4)
            {
                signal_control = 2;
            }
            else
            {
                signal_control = 3;
            }
        }

    }
    #endregion

    #region


    if (t % 60 == 0)
    {
```

```
        double cs = random.Next(3, 6);
        int born_number = (int)(Math.Floor(cs));
        int dspeed = pm.NormalRandom();
        Console.WriteLine(dspeed);
        ki = born_number;
        for (int ii = 0; ii < born_number; ii++)
        {
            Location[ii * (dspeed + L_veh)] = 1;
            Velocity[ii * (dspeed + L_veh)] = dspeed;
            Desired_speed[ii * (dspeed + L_veh)] = dspeed;
        }
    }
    else
        ki = 0;


    #endregion

    #region

    int N = 0;
    for (int j = 0; j < Road_Length; j++)
    {
        if (Location[j] == 1)
        {
            N++;
        }
    }
    if (N == 0)
    {
        continue;
    }
    if (t == 3000)
    {
        t = 3000;

    }
    int[] Location_1 = new int[N];
    int[] Num_Location_1 = new int[N];

#endregion

    #region
    if (tt > green && tt < green + 5)
    {
        for (int j = 0; j < Road_Length; j++)
        {
            stgo_1[j] = stgo[j];
        }
        for (int j = 0; j < Road_Length; j++)
        {
            stgo[j] = 0;
        }
        for (int j = 0; j < Road_Length - 1; j++)
        {
            stgo[j + ki] = stgo_1[j];
        }
    }
```

```
#endregion

#region
int[] Distance = new int[N];
int[] Num_Location = new int[N];
int[] Num_Velocity = new int[N];
int[] Virtual_Velocity = new int[N];
int[] kkk = new int[N];

int Number = 1;
int i = 0;
while (i < Road_Length)
{
    int[] a = new int[2];
    int c = 0;
    while (i < Road_Length && c < 2)
    {
        if (Location[i] == 1)
        {
            a[c] = i;
            c++;
        }
        i++;
    }
    if (c == 2)
    {
        i--;
        c = 0;
        Distance[Number - 1] = a[1] - a[0] - L_veh;
        Num_Location[Number - 1] = a[0];
        Num_Velocity[Number - 1] = Velocity[a[0]];
        Number++;
    }
    else
    {
        if (a[0] >= 4995)
        {
            Distance[Number - 1] = Vmax;
            switch (signal_control)
            {
                case 1:
                    Distance[Number - 1] = Vmax;
                    break;
                case 2:
                    if (stgo[Number - 1] == 1)
                        Distance[Number - 1] = Vmax;
                    else
                        Distance[Number - 1] = 0;
                    break;
                case 3:
                    if (tt <= green + 5)
                        Distance[Number - 1] = Vmax;
                    else
                        Distance[Number - 1] = 0;
                    break;
            }
        }
        else
```

```
                {
                    if (signal_control == 1)
                    {
                        Distance[Number - 1] = Vmax;
                    }
                    else
                    {
                        if (signal_control == 2)
                        {
                            if (stgo[Number - 1] == 1)
                                Distance[Number - 1] = Vmax;
                            else
                                Distance[Number - 1] = Road_Length - a[0]
- L_veh;

                        }
                        else
                        {
                            Distance[Number - 1] = Road_Length - a[0] -
L_veh;

                        }

                    }
                }
                Num_Location[Number - 1] = a[0];
                Num_Velocity[Number - 1] = Velocity[a[0]];
                break;
            }
        }
        for (int j = 0; j < Number; j++)
        {
            if (Distance[j] < 0)
            {
                error++;
            }
        }
        #endregion

        for (int j = 0; j < Number; j++)
        {
            if (Distance[j] < 0)
            {
                error++;
            }
        }

        for (int j = 0; j < Number - 1; j++)
        {
            if (Num_Velocity[j] - Num_Velocity[j + 1] > Distance[j])
            {
                error++;
            }
        }



        #region
        if (t > TT1)
        {
```

```
                            for (int j = ki + kii; j < Number - 1; j++)
                            {
                                if (Num_Velocity[j] < Vmax && Distance[j] == 0)
                                {
                                    if (Velocity_2[j + 1 - ki - kii] >= 0)
                                    {
                                        if (Velocity_2[j - ki - kii] - Num_Velocity[j] >
2)
                                        {

                                            if (Num_Velocity[j + 1] == 0)
                                            {
                                                DS1[t - TT1 - 1] += 1;
                                            }
                                            else
                                            {
                                                DS2[t - TT1 - 1] += 1;
                                            }
                                            if (t_l11 < 5000)
                                            {
                                                t_l1[0, t_l11] = tt;
                                                t_l1[1, t_l11] = Num_Location[j];
                                                t_l1[2, t_l11] = alpha;
                                                t_l1[3, t_l11] =
Desired_speed[Num_Location[j]];
                                                t_l11++;
                                            }
                                        }
                                    }
                                }

                            }
                            DS1[t - TT1 - 1] /= (Number - 1);
                            DS2[t - TT1 - 1] /= (Number - 1);
                        }
                        #endregion

                        Program pp = new Program();

                        #region

                        if (tt == green + 1)
                        {
                            int Num_stgo_error = 0;
                            int Num_pdcar = 0;

                            for (int j = 0; j < Number; j++)
                            {
                                if (Num_Location[j] > 4900 && Num_Location[j] < 4995)
                                {
                                    Num_pdcar++;
                                    if (Num_Location[j] >= 4940)
                                        stgo[j] = 1;
                                    else
                                        stgo[j] = 2;
                                    if (stgo[j] == 1 && ((5000 - Num_Location[j]) > 4 *
Num_Velocity[j]))
                                        Num_stgo_error++;
```

```
                                }
                                else
                                {
                                    stgo[j] = 2;
                                }

                            if (Num_pdcar>0)
                            {
                                stgo_error[stgo_para] = (double)Num_stgo_error /
Num_pdcar;
                                stgo_para++;
                            }
                        }
                        #endregion

                        #region
                        int[] r = new int[N];
                        for (int j = 0; j < Number; j++)
                        {
                            if (t == t0 || Number < 3)
                            {
                                r[j] = 1;
                            }
                            else
                            {
                                if (j < Number - 2)
                                {
                                    if ((Num_Velocity[j] <= Num_Velocity[j + 1] &&
Num_Velocity[j + 1] <= Num_Velocity[j + 2]) || Num_Velocity[j + 2] >=
Math.Max(Desired_speed[Num_Location[j + 2]], 19))
                                    {
                                        r[j] = 0;
                                    }
                                    else
                                    {
                                        r[j] = 1;
                                    }
                                }
                                else
                                {
                                    if (j == Number - 2)
                                    {
                                        if (Num_Velocity[j] < Num_Velocity[j + 1])
                                        {
                                            r[j] = 0;
                                        }
                                        else
                                        {
                                            r[j] = 1;
                                        }
                                    }
                                    else
                                    {
                                         r[j] = 0;
                                    }
                                }
                            }
```

```
                            if (Num_Location[j] > 4914 && tt > 29)
                                r[j] = 1;
                        }

                        #endregion
                        #region
                        int[] Delta = new int[N];
                        int[] t_f = new int[N];
                        int[] t_l = new int[N];
                        int[] aa = new int[N];
                        for (int j = 0; j < Number; j++)
                        {
                            t_l[j] = (int)Math.Floor(r[j] * (double)(Num_Velocity[j]) / D
+ (1 - r[j]) * Math.Min((double)(Num_Velocity[j]) / D, t_safe));

                            Delta[j] = Num_Velocity[j] * 2;
                        }
                        #endregion
                        #region
                        int[] C_n = new int[N];
                        int[] C_n1 = new int[N];
                        int[] left = new int[N];
                        int[] C_n_limit = new int[N];
                        int VC_n;
                        for (int j = 0; j < Number; j++)
                        {
                            if (j < Number - 1)
                            {
                                if (Num_Location[j] >= 4900)
                                {
                                    VC_n = Desired_speed[Num_Location[j]];
                                }
                                else
                                {
                                    VC_n = Desired_speed[Num_Location[j]];
                                }
                                for (C_n[j] = Math.Max(Num_Velocity[j] - 2, 0); C_n[j] <=
Math.Min(Num_Velocity[j] + 2, VC_n); C_n[j]++)
                                {
                                    t_f[j] = (int)Math.Ceiling(r[j] * (double)(C_n[j]) /
D + (1 - r[j]) * Math.Max(0, Math.Min((double)(C_n[j]) / D, t_safe) - 1));
                                    if (t_f[j] == 0)
                                    {
                                        aa[j] = 0;
                                    }
                                    else
                                    {
                                        aa[j] = 1;
                                    }
                                    if (C_n[j] % D == 0)
                                    {
                                        if ((t_f[j] + 1) * C_n[j] > (int)(Distance[j] +
Num_Velocity[j + 1] * t_l[j + 1] - 0.5 * (t_l[j + 1] + 1) * t_l[j + 1] * D + 0.5 *
(t_f[j] + 1) * t_f[j] * D))
                                        {
                                            C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                            break;
                                        }
```

```
                                    else
                                    {
                                        C_n1[j] = C_n[j];
                                        continue;
                                    }
                                }
                                else
                                {
                                    if ((t_f[j] + 1) * C_n[j] > (int)(Distance[j] +
Num_Velocity[j + 1] * t_l[j + 1] - 0.5 * (t_l[j + 1] + 1) * t_l[j + 1] * D + 0.5 *
(t_f[j] - 1) * t_f[j] * D + aa[j] * (2 * t_f[j] - 1)))
                                    {
                                        C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                        break;
                                    }
                                    else
                                    {
                                        C_n1[j] = C_n[j];
                                        continue;
                                    }
                                }
                            }
                        }
                        else
                        {
                            if (Num_Location[j] >= 4900)
                            {
                                VC_n = Desired_speed[Num_Location[j]];
                            }
                            else
                            {
                                VC_n = Desired_speed[Num_Location[j]];
                            }
                            if (signal_control == 1)
                            {
                                C_n1[j] = Num_Velocity[j] + 1;
                            }
                            else
                            {
                                if (signal_control == 2)
                                {
                                  if (Distance[j] == Vmax && Num_Location[j] >= 4995)
                                    {
                                        C_n1[j] = Num_Velocity[j] + 1;
                                    }
                                    else
                                    {
                                        for (C_n[j] = Math.Max(Num_Velocity[j] - 2,
0); C_n[j] <= Math.Min(Num_Velocity[j] + 2, VC_n); C_n[j]++)
                                        {
                                            t_f[j] = (int)Math.Ceiling(r[j] *
(double)(C_n[j]) / D + (1 - r[j]) * Math.Max(0, Math.Min((double)(C_n[j]) / D, t_safe) -
1));
                                            if (t_f[j] == 0)
                                            {
                                                aa[j] = 0;
                                            }
                                            else
```

```
                                                {
                                                    aa[j] = 1;
                                                }
                                                if (C_n[j] % D == 0)
                                                {
                                                    if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] + 1) * t_f[j] * D))
                                                    {
                                                        C_n1[j] = Math.Max(C_n[j] - 1,
0);
                                                        break;
                                                    }
                                                    else
                                                    {
                                                        C_n1[j] = C_n[j];
                                                        continue;
                                                    }
                                                }
                                                else
                                                {
                                                    if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] - 1) * t_f[j] * D + aa[j] * (2 * t_f[j] - 1)))
                                                    {
                                                        C_n1[j] = Math.Max(C_n[j] - 1,
0);
                                                        break;
                                                    }
                                                    else
                                                    {
                                                        C_n1[j] = C_n[j];
                                                        continue;
                                                    }
                                                }
                                            }
                                        }
                                    }
                                    else
                                    {
                                        for (C_n[j] = Math.Max(Num_Velocity[j] - 2, 0);
C_n[j] <= Math.Min(Num_Velocity[j] + 2, VC_n); C_n[j]++)
                                        {
                                            t_f[j] = (int)Math.Ceiling(r[j] *
(double)(C_n[j]) / D + (1 - r[j]) * Math.Max(0, Math.Min((double)(C_n[j]) / D, t_safe) -
1));
                                            if (t_f[j] == 0)
                                            {
                                                aa[j] = 0;
                                            }
                                            else
                                            {
                                                aa[j] = 1;
                                            }
                                            if (C_n[j] % D == 0)
                                            {
                                                if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] + 1) * t_f[j] * D))
                                                {
                                                    C_n1[j] = Math.Max(C_n[j] - 1, 0);
```

```
                                            break;
                                        }
                                        else
                                        {
                                            C_n1[j] = C_n[j];
                                            continue;
                                        }
                                    }
                                    else
                                    {
                                        if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] - 1) * t_f[j] * D + aa[j] * (2 * t_f[j] - 1)))
                                        {
                                            C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                            break;
                                        }
                                        else
                                        {
                                            C_n1[j] = C_n[j];
                                            continue;
                                        }
                                    }
                                }
                            }
                        }
                    }

                    #endregion


                    #region
                    int[] ad = new int[5000];
                    if (tt > green && tt < green + 5)
                    {
                        int cc = 0;
                        for (int j = Number - 1; j > 0; j--)
                        {
                            if (stgo[j] == 2)
                                cc++;
                            if (Num_Location[j] > 4920 && Num_Location[j] < 4995 &&
(j == Number - 1 || (stgo[j] == 2 && cc == 0)))
                            {
                                {
                                  double gap = (double)Distance[j] / Num_Velocity[j];
                                    int stbar = 4995 - Num_Location[j];
                                    ad[j] = pp.AccDec(gap, stgo[j], stbar,
Num_Velocity[j]);
                                }
                            }
                        }
                    }

                    #endregion

                    #region
                    int[] Num_Velocity_Last = new int[N];
                    for (int j = 0; j < Number; j++)
```

```
                        {
                            switch (ad[j])
                            {
                                case 0:
                                    Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Num_Velocity[j] + 1,
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                    break;
                                case 1:
                                    Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Num_Velocity[j] + 1,
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                    break;
                                case 2:
                                    Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Math.Max(Num_Velocity[j] - 1, 0),
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                    break;
                                case 3:
                                    Num_Velocity_Last[j] = Num_Velocity[j];
                                    break;
                                case 4:
                                    Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Math.Max(Num_Velocity[j] - 1, 0),
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                    break;
                                case 5:
                                    Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Math.Max(Num_Velocity[j] - 2, 0),
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                    break;
                            }
                        }

                       #endregion

                        #region

                        for (int j = 0; j < Number; j++)
                        {
                            double sj = random.Next(0, 999);
                            sj /= 1000;
                            if (((signal_control == 1 || signal_control == 2) &&
Num_Location[j] >= 4995) || kkk[Number - 1] == 1)
                            {
                                continue;
                            }
                            if (sj < 0.16)
                            {
                                Num_Velocity_Last[j] = Math.Max(0,
Math.Max(Num_Velocity[j] - D, Num_Velocity_Last[j] - 1));
                            }
                        }
                        for (int j = 0; j < Number; j++)
                        {
                            Num_Velocity[j] = Num_Velocity_Last[j];
                        }
                        #endregion
```

```
for (int j = Number - 1; j >= 0; j--)
{
    if (j < Number - 1)
    {
        if (Num_Velocity[j] - Num_Velocity[j + 1] > Distance[j])
        {
            Num_Velocity[j] = Num_Velocity[j + 1] + Distance[j];
        }
    }
    else
    {
        if (Num_Velocity[j] > Distance[j])
        {
            if (j == Number - 1 && Distance[j] < 0)
                Num_Velocity[j] = 0;
            else
                Num_Velocity[j] = Distance[j];
        }
    }
}

for (int j = 0; j < Number; j++)
{
    if (Num_Velocity[j] < 0)
    {
        error++;
    }
}

#region
k = 0;
kii = ki;
if (t >= TT1 - 2)
{
    for (int j = 0; j < Number; j++)
    {
        Velocity_2[j] = Velocity_1[j];
    }
}
for (int j = 0; j < Number; j++)
{
    Num_Location_1[j] = Num_Location[j] + Num_Velocity[j];
    Velocity_1[j] = Num_Velocity[j];

    if (j <= Number - 1 && j >= ki)
    {
        if (t > TT1 && Velocity_2[j - ki] - Num_Velocity[j] > 2)
        {
            Slam_Brake1[t - TT1 - 1]++;

            if (t_122 < 5000)
            {
                t_12[0, t_122] = tt;
                t_12[1, t_122] = Num_Location[j];
                if (Num_Location[j] < 4900)
                    error++;
                t_12[2, t_122] = alpha;
```

```
                                    t_l2[3, t_l11] = Desired_speed[Num_Location[j]];
                                    t_l22++;
                                }
                            }
                        }
                        if (Num_Location_1[j] >= Road_Length)
                        {
                            k++;
                            if (t >= TT1)
                            {
                                Flux[(int)Math.Round(100 * (alpha - 0.01))]++;
                            }
                        }

                    }
                    if (t > TT1)
                    {
                        Slam_Brake1[t - TT1 - 1] /= Number;
                    }
                    for (int j = 0; j < Number - k; j++)
                    {
                        Desired_speed_1[Num_Location_1[j]] =
Desired_speed[Num_Location[j]];
                    }
                    for (int j = 0; j < Number; j++)
                    {
                        Num_Location[j] = 0;
                    }
                    for (int j = 0; j < Number - k; j++)
                    {
                        Num_Location[j] = Num_Location_1[j];
                    }
                    for (int j = 0; j < Road_Length; j++)
                    {
                        Location[j] = 0;
                        Velocity[j] = 0;
                        Desired_speed[j] = 0;
                    }
                    for (int j = 0; j < Road_Length; j++)
                    {
                        Desired_speed[j] = Desired_speed_1[j];
                    }
                    for (int j = 0; j < Number - k; j++)
                    {
                        Location[Num_Location[j]] = 1;
                        Velocity[Num_Location[j]] = Num_Velocity[j];
                    }
                    #endregion



                    if (Number > 5)
                    {
                        for (int j = 0; j < Number - Math.Max(1, k); j++)
                        {
                            if (Num_Location[j] >= Num_Location[j + 1] &&
Num_Location[j + 1] != 0)
                            {
```

```
                                error++;
                            }
                        }
                    }
                }

                #region

                for (int j = 0; j < TT; j++)
                {
                    DSZ1[(int)Math.Round(100 * (alpha - 0.01))] += DS1[j];
                    DSZ2[(int)Math.Round(100 * (alpha - 0.01))] += DS2[j];
                    Slam_Brake[(int)Math.Round(100 * (alpha - 0.01))] +=
Slam_Brake1[j];
                }
                DSZ1[(int)Math.Round(100 * (alpha - 0.01))] /= TT;
                DSZ2[(int)Math.Round(100 * (alpha - 0.01))] /= TT;
                Slam_Brake[(int)Math.Round(100 * (alpha - 0.01))] /= TT;

                #endregion
            }
            for (int j = 0; j < 100; j++)
            {
                DSZ11[j] += DSZ1[j];
                DSZ22[j] += DSZ2[j];
                Flux11[j] += Flux[j];
                Slam_Brake11[j] += Slam_Brake[j];
                if (DSZ11[j] == 0 || DSZ22[j] == 0)
                {
                    error++;
                }

            }

        }
        for (int j = 0; j < 100; j++)
        {
            DSZ11[j] /= CC;
            DSZ22[j] /= CC;
            Flux11[j] /= CC;
            Slam_Brake11[j] /= CC;
        }
        FileStream dsz1 = File.Create("E:\\DSZ1.txt");
        FileStream dsz2 = File.Create("E:\\DSZ2.txt");
        StreamWriter swdsz1 = new StreamWriter((System.IO.Stream)dsz1);
        StreamWriter swdsz2 = new StreamWriter((System.IO.Stream)dsz2);
        for (int j = 0; j < 100; j++)
        {
            swdsz1.Write("{0} ", DSZ11[j]);
            swdsz2.Write("{0} ", DSZ22[j]);
        }
        swdsz1.Close();
        swdsz2.Close();
        FileStream flow = File.Create("E:\\FLUX.txt");
        StreamWriter swflow = new StreamWriter((System.IO.Stream)flow);
        for (int j = 0; j < 100; j++)
        {
            swflow.Write("{0} ", Flux11[j]);
```

```
        }
        swflow.Close();
        FileStream brake = File.Create("E:\\Brake1.txt");
        StreamWriter swbrake = new StreamWriter((System.IO.Stream)brake);
        for (int j = 0; j < 100; j++)
        {
            swbrake.Write("{0} ", Slam_Brake11[j]);
        }
        swbrake.Close();
        FileStream st1 = File.Create("E:\\tl1.txt");
        StreamWriter stw1 = new StreamWriter((System.IO.Stream)st1);
        FileStream st2 = File.Create("E:\\tl2.txt");
        StreamWriter stw2 = new StreamWriter((System.IO.Stream)st2);
        for (int jj = 0; jj < 4; jj++)
        {
            for (int j = 0; j < 5000; j++)
            {
                stw1.Write("{0} ", t_l1[jj, j]);
                stw2.Write("{0} ", t_l2[jj, j]);
            }
            stw1.Write("\n");
            stw2.Write("\n");
        }
        stw1.Close();
        stw2.Close();
        FileStream stgo_Error = File.Create("E:\\stgo_error.txt");
        StreamWriter sger = new StreamWriter((System.IO.Stream)stgo_Error);
        for (int j = 0; j < 10000; j++)
        {
            sger.Write("{0} ", stgo_error[j]);
        }
        sger.Close();

                    #endregion

        Console.WriteLine("OVER");
        Console.ReadLine();
    }
    public int StopGo(int di, int sp, bool lf)
    {

        #region
        Random aa = new Random();
        if (di > 4941)
        {
            double cc = aa.Next(0, 999) / 1000;
            if (cc < 0.191)
                return 2;
            else
                return 1;
        }
        else
            if (di < 4924)
            {

                double cc = aa.Next(0, 999) / 1000;
                if (cc < 0.922)
                    return 2;
```

```
            else
                return 1;
        }
        else
            if (sp < 15)
            {
                double cc = aa.Next(0, 999) / 1000;
                if (cc < 0.604)
                    return 2;
                else
                    return 1;
            }
            else
                if (lf)
                {
                    double cc = aa.Next(0, 999) / 1000;
                    if (cc < 0.488)
                        return 2;
                    else
                        return 1;
                }
                else
                {
                    double cc = aa.Next(0, 999) / 1000;
                    if (cc < 59.6)
                        return 2;
                    else
                        return 1;
                }
    #endregion
}

public int AccDec(double gap, int stopgo, int Dtostopbar, int speed)
{
    Random pa = new Random();
    if (stopgo == 1)
    {
        if (gap <= 3)
        {
            double aa = pa.Next(0, 1000);
            aa /= 1000;
            if (aa < 0.7686)
                return 2;
            else
                return 3;
        }
        else if (gap >= 3.6)
        {
            double aa = pa.Next(0, 1000);
            aa /= 1000;
            if (aa < 0.07)
                return 1;
            else
                return 3;
        }
        else
            return 3;
    }
```

```
            else
            {
                int d1, d2;
                d1 = speed * (speed - 1) / 2;
                if (speed % 2 == 0)
                    d2 = speed * (speed - 2) / 4;
                else
                    d2 = (speed - 1) * (speed - 1) / 4;
                if (Dtostopbar > d1)
                {
                    if (Dtostopbar - speed > d1)
                        return 0;
                    else if (Dtostopbar - speed - 1 > (speed - 1) * (speed - 2) / 2)
                    {
                        return 4;
                    }
                    else
                    {
                        return 5;
                    }
                }
                else if (Dtostopbar > d2)
                    return 5;
                else
                    return 5;
            }
        }

        public double ST_GO_LogitRegression(bool lead, int spe, int dis)
        {
            int x1 = 0, x2 = 0, x3 = 0, x4 = 0, x5 = 0, x6 = 0;
            if (lead)
                x1 = 1;

            if (spe > 14 && spe < 16)
                x2 = 1;
            else
                if (spe < 20)
                    x3 = 1;
            if (dis > 57 && dis < 79)
                x4 = 1;
            else
                if (dis >= 79 && dis < 88)
                    x5 = 1;
                else
                    if (dis >= 88 && dis < 98)
                        x6 = 1;

            double y = Math.Exp(-0.1945 + 0.9350 * x1 + 1.4994 * x2 + 3.2820 * x3 -
2.4108 * x4 - 4.5557 * x5 - 5.2498 * x6) /
                (1 + Math.Exp(-0.1945 + 0.9350 * x1 + 1.4994 * x2 + 3.2820 * x3 - 2.4108
* x4 - 4.5557 * x5 - 5.2498 * x6));
            return y;
        }

        public double AverageRandom(double min, double max)
        {
            double aa = random.Next(0, 1001);
```

```
            aa = aa / 1000;
            return min + aa * (max - min);
        }

        public double Normal(double x, double mu, double sigma)
        {
            return 1.0 / Math.Sqrt(2 * Math.PI * sigma) * Math.Exp(-1 * (x - mu) * (x -
mu) / (2 * sigma * sigma));
        }

        public int NormalRandom()
        {
            double x;
            double y;
            double dScope;
            do
            {
                x = AverageRandom(34, 66);
                dScope = Normal(x, 50, 5);
                y = AverageRandom(0, 0.14);
            }
            while (y >= dScope);
            x = (int)Math.Round(x * 0.44704 / 1.5, 0);
            return (int)x;
        }
    }
}
```

APPENDIX D C# CODE FOR THE PMAIC SCENARIO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
namespace Amber_flash_add_Pavement_marking
{
    class Program
    {
        Random random = new Random();
        static void Main(string[] args)
        {
            Random random = new Random();
            int Road_Length = 5000;
            int Vmax = 20;
            int Vfast = 20;
            double[] P = { 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 };
            int t0 = 0;
            int TT1 = 10000;
            int TT = 5000;
            double alpha;
            int L_veh = 5;
            int green = 25;
            int C = 60;
            int red = 25;
            int ki;
            int kii = 0;
            int error = 0;
            int D = 2;
            int t_safe = 3;
            int Vlimit = 15;
            double[,] t_l1 = new double[4, 5000];
            double[,] t_l2 = new double[4, 5000];
            int t_l11 = 0;
            int t_l22 = 0;
            double[] DSZ11 = new double[100];
            double[] DSZ22 = new double[100];
            int[] Flux11 = new int[100];
            double[] Slam_Brake11 = new double[100];
            double[] stgo_error = new double[10000];
            int stgo_para = 0;
            int CC = 1;
            int k = 0;
            for (int mm = 0; mm < CC; mm++)
            {
                double[] DSZ1 = new double[100];
                double[] DSZ2 = new double[100];
                int[] Flux = new int[100];
                double[] Slam_Brake = new double[100];
                int[,] Time_Location = new int[300, Road_Length];
                int[,] ArrTL = new int[240, 2000];

                int M = 0;
                int Dis = 0;
                int para_signal = 0;
                Program pm = new Program();
                for (alpha = 0.01; alpha < 0.401; alpha += 0.01)
                {
```

```
#region
int[] Location = new int[Road_Length];
int[] Velocity = new int[Road_Length];
double[] DS1 = new double[TT];
double[] DS2 = new double[TT];
double[] Slam_Brake1 = new double[TT];
int[] Desired_speed = new int[5000];
int[] Desired_speed_1 = new int[5000];
int[] Velocity_1 = new int[Road_Length];
int[] stgo_1 = new int[Road_Length];
int[] Velocity_2 = new int[Road_Length];
int[] stgo = new int[5000];
int signal_control = 1;
int tt = 0;
#endregion
for (int t = t0; t < (TT1 + TT); t++)
{
    if (tt == green + 5)
    {
        for (int ii = 0; ii < Road_Length; ii++)
        {
            stgo[ii] = 0;
        }
    }

    #region
    if (t > t0)
    {
        if (tt >= C)
        {
            tt = tt - C;
        }
        tt = tt + 1;
        if (tt <= green)
        {
            signal_control = 1;
        }
        else
        {
            if (tt <= green + 4)
            {
                signal_control = 2;
            }
            else
            {
                signal_control = 3;
            }
        }

    }
    #endregion

    #region
    if (t % 60 == 0)
    {
        double cs = random.Next(3, 6);
        int born_number = (int)(Math.Floor(cs));
        int dspeed = pm.NormalRandom();
```

```
        ki = born_number;
        for (int ii = 0; ii < born_number; ii++)
        {
            Location[ii * (dspeed + L_veh)] = 1;
            Velocity[ii * (dspeed + L_veh)] = dspeed;
            Desired_speed[ii * (dspeed + L_veh)] = dspeed;
        }
    }
    else
        ki = 0;
#endregion
    #region
    int N = 0;
    for (int j = 0; j < Road_Length; j++)
    {
        if (Location[j] == 1)
        {
            N++;
        }
    }
    if (N == 0)
    {
        continue;
    }
    if (t == 3000)
    {
        t = 3000;

    }
    int[] Location_1 = new int[N];
    int[] Num_Location_1 = new int[N];
    #endregion
    #region stopgo
    if (tt > green - 5 && tt < green + 5)
    {
        for (int j = 0; j < Road_Length; j++)
        {
            stgo_1[j] = stgo[j];
        }
        for (int j = 0; j < Road_Length; j++)
        {
            stgo[j] = 0;
        }
        for (int j = 0; j < Road_Length - 1; j++)
        {
            stgo[j + ki] = stgo_1[j];
        }
    }
    #endregion

    #region

    int[] Distance = new int[N];
    int[] Num_Location = new int[N];
    int[] Num_Velocity = new int[N];
    int[] Virtual_Velocity = new int[N];
    int[] kkk = new int[N];
```

```
int Number = 1;
int i = 0;
while (i < Road_Length)
{
    int[] a = new int[2];
    int c = 0;
    while (i < Road_Length && c < 2)
    {
        if (Location[i] == 1)
        {
            a[c] = i;
            c++;
        }
        i++;
    }
    if (c == 2)
    {
        i--;
        c = 0;
        Distance[Number - 1] = a[1] - a[0] - L_veh;
        Num_Location[Number - 1] = a[0];
        Num_Velocity[Number - 1] = Velocity[a[0]];
        Number++;
    }
    else
    {
        if (a[0] >= 4995)
        {
            Distance[Number - 1] = Vmax;
            switch (signal_control)
            {
                case 1:
                    Distance[Number - 1] = Vmax;
                    break;
                case 2:
                    if (stgo[Number - 1] == 1)
                        Distance[Number - 1] = Vmax;
                    else
                        Distance[Number - 1] = 0;
                    break;
                case 3:
                    if (tt <= green + 5)
                        Distance[Number - 1] = Vmax;
                    else
                        Distance[Number - 1] = 0;
                    break;
            }
        }
        else
        {
            if (signal_control == 1)
            {
                Distance[Number - 1] = Vmax;
            }
            else
            {
                if (signal_control == 2)
                {
```

```
                    if (stgo[Number - 1] == 1)
                        Distance[Number - 1] = Vmax;
                    else
                Distance[Number - 1] = Road_Length - a[0] - L_veh;
                 }
                 else
                 {
                Distance[Number - 1] = Road_Length - a[0] - L_veh;
                 }

            }
        }
        Num_Location[Number - 1] = a[0];
        Num_Velocity[Number - 1] = Velocity[a[0]];
        break;
    }
}
for (int j = 0; j < Number; j++)
{
    if (Distance[j] < 0)
    {
        error++;
    }
}
#endregion

for (int j = 0; j < Number; j++)
{
    if (Distance[j] < 0)
    {
        error++;
    }
}

for (int j = 0; j < Number - 1; j++)
{
    if (Num_Velocity[j] - Num_Velocity[j + 1] > Distance[j])
    {
        error++;
    }
}



#region
if (t > TT1)
{
    for (int j = ki + kii; j < Number - 1; j++)
    {
            if (Num_Velocity[j] < Vmax && Distance[j] == 0)
        {
            if (Velocity_2[j + 1 - ki - kii] >= 0)
            {
                if (Velocity_2[j - ki - kii] - Num_Velocity[j] >
2)

                {

                    if (Num_Velocity[j + 1] == 0)
```

```
                                        {
                                            DS1[t - TT1 - 1] += 1;
                                        }
                                        else
                                        {
                                            DS2[t - TT1 - 1] += 1;
                                        }
                                        if (t_l11 < 5000)
                                        {
                                            t_l1[0, t_l11] = tt;
                                            t_l1[1, t_l11] = Num_Location[j];
                                            t_l1[2, t_l11] = alpha;
                                            t_l1[3, t_l11] =
Desired_speed[Num_Location[j]];
                                            t_l11++;
                                        }
                                    }
                                }
                            }

                        }
                        DS1[t - TT1 - 1] /= (Number - 1);
                        DS2[t - TT1 - 1] /= (Number - 1);
                    }
                    #endregion

                    Program pp = new Program();
                    #region
                    if (tt == green - 2)
                    {
                        for (int j = 0; j < Number; j++)
                        {
                            if (Num_Location[j] > 4900 && Num_Location[j] < 4940 &&
Num_Velocity[j] < 15)
                            {
                                stgo[j] = 2;
                            }
                        }
                    }
                    if (tt == green + 1)
                    {
                        for (int j = 0; j < Number; j++)
                        {
                            if (Num_Location[j] > 4900 && Num_Location[j] < 4995 &&
stgo[j] == 0)
                            {
                                if (Num_Location[j] >= 4940)
                                    stgo[j] = 1;
                                else
                                    stgo[j] = 2;

                            }
                            else
                            {
                                stgo[j] = 2;
                            }
                        }
```

```
                        }

                        #endregion
                        #region
                        if (tt == green + 2)
                        {
                            int Num_stgo_error = 0;
                            int Num_pdcar = 0;
                            for (int j = Number - 1; Num_Location[j] > 4900; j--)
                            {
                                Num_pdcar++;
                    if (stgo[j] == 1 && ((5000 - Num_Location[j]) > 4 * Num_Velocity[j]))
                                    Num_stgo_error++;
                            }
                            if (Num_pdcar > 0)
                            {
                                stgo_error[stgo_para] = (double)Num_stgo_error /
Num_pdcar;
                                stgo_para++;
                            }
                        }

                        #endregion
                        #region

                        int[] r = new int[N];
                        for (int j = 0; j < Number; j++)
                        {
                            if (t == t0 || Number < 3)
                            {
                                r[j] = 1;
                            }
                            else
                            {
                                if (j < Number - 2)
                                {
                                    if ((Num_Velocity[j] <= Num_Velocity[j + 1] &&
Num_Velocity[j + 1] <= Num_Velocity[j + 2]) || Num_Velocity[j + 2] >=
Math.Max(Desired_speed[Num_Location[j + 2]], 19))
                                    {
                                        r[j] = 0;
                                    }
                                    else
                                    {
                                        r[j] = 1;
                                    }
                                }
                                else
                                {
                                    if (j == Number - 2)
                                    {
                                        if (Num_Velocity[j] < Num_Velocity[j + 1])
                                        {
                                            r[j] = 0;
                                        }
                                        else
                                        {
                                            r[j] = 1;
```

```
                                    }
                                }
                                else
                                {
                                    r[j] = 0;
                                                                                    }
                                }
                            }
                            if (Num_Location[j] > 4914 && tt > 29)
                                r[j] = 1;
                        }

                        #endregion
                        #region
                        int[] Delta = new int[N];
                        int[] t_f = new int[N];
                        int[] t_l = new int[N];
                        int[] aa = new int[N];
                        for (int j = 0; j < Number; j++)
                        {
                            t_l[j] = (int)Math.Floor(r[j] * (double)(Num_Velocity[j]) / D
+ (1 - r[j]) * Math.Min((double)(Num_Velocity[j]) / D, t_safe));

                            Delta[j] = Num_Velocity[j] * 2;
                        }
                        #endregion

                        #region
                        int[] C_n = new int[N];
                        int[] C_n1 = new int[N];
                        int[] left = new int[N];
                        int[] C_n_limit = new int[N];
                        int VC_n;
                        for (int j = 0; j < Number; j++)
                        {
                            if (j < Number - 1)
                            {
                                if (Num_Location[j] >= 4900)
                                {
                                    VC_n = Desired_speed[Num_Location[j]];
                                }
                                else
                                {
                                    VC_n = Desired_speed[Num_Location[j]];
                                }
                                for (C_n[j] = Math.Max(Num_Velocity[j] - 2, 0); C_n[j] <=
Math.Min(Num_Velocity[j] + 2, VC_n); C_n[j]++)
                                {
                                    t_f[j] = (int)Math.Ceiling(r[j] * (double)(C_n[j]) /
D + (1 - r[j]) * Math.Max(0, Math.Min((double)(C_n[j]) / D, t_safe) - 1));
                                    if (t_f[j] == 0)
                                    {
                                        aa[j] = 0;
                                    }
                                    else
                                    {
                                        aa[j] = 1;
                                    }
```

```
                                    if (C_n[j] % D == 0)
                                    {
                                        if ((t_f[j] + 1) * C_n[j] > (int)(Distance[j] +
Num_Velocity[j + 1] * t_l[j + 1] - 0.5 * (t_l[j + 1] + 1) * t_l[j + 1] * D + 0.5 *
(t_f[j] + 1) * t_f[j] * D))
                                        {
                                            C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                            break;
                                        }
                                        else
                                        {
                                            C_n1[j] = C_n[j];
                                            continue;
                                        }
                                    }
                                    else
                                    {
                                        if ((t_f[j] + 1) * C_n[j] > (int)(Distance[j] +
Num_Velocity[j + 1] * t_l[j + 1] - 0.5 * (t_l[j + 1] + 1) * t_l[j + 1] * D + 0.5 *
(t_f[j] - 1) * t_f[j] * D + aa[j] * (2 * t_f[j] - 1)))
                                        {
                                            C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                            break;
                                        }
                                        else
                                        {
                                            C_n1[j] = C_n[j];
                                            continue;
                                        }
                                    }
                                }
                            }
                            else
                            {
                                if (Num_Location[j] >= 4900)
                                {
                                    VC_n = Desired_speed[Num_Location[j]];
                                }
                                else
                                {
                                    VC_n = Desired_speed[Num_Location[j]];
                                }
                                if (signal_control == 1)
                                {
                                    C_n1[j] = Num_Velocity[j] + 1;
                                }
                                else
                                {
                                    if (signal_control == 2 && stgo[j] != 2)
                                    {
                                      if (Distance[j] == Vmax && Num_Location[j] >= 4995)
                                        {
                                            C_n1[j] = Num_Velocity[j] + 1;
                                        }
                                        else
                                        {
                                            for (C_n[j] = Math.Max(Num_Velocity[j] - 2,
0); C_n[j] <= Math.Min(Num_Velocity[j] + 2, VC_n); C_n[j]++)
```

```
                                        {
                                            t_f[j] = (int)Math.Ceiling(r[j] *
(double)(C_n[j]) / D + (1 - r[j]) * Math.Max(0, Math.Min((double)(C_n[j]) / D, t_safe) -
1));
                                            if (t_f[j] == 0)
                                            {
                                                aa[j] = 0;
                                            }
                                            else
                                            {
                                                aa[j] = 1;
                                            }
                                            if (C_n[j] % D == 0)
                                            {
                                                if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] + 1) * t_f[j] * D))
                                                {
                                                    C_n1[j] = Math.Max(C_n[j] - 1,
0);
                                                    break;
                                                }
                                                else
                                                {
                                                    C_n1[j] = C_n[j];
                                                    continue;
                                                }
                                            }
                                            else
                                            {
                                                if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] - 1) * t_f[j] * D + aa[j] * (2 * t_f[j] - 1)))
                                                {
                                                    C_n1[j] = Math.Max(C_n[j] - 1,
0);
                                                    break;
                                                }
                                                else
                                                {
                                                    C_n1[j] = C_n[j];
                                                    continue;
                                                }
                                            }
                                        }
                                    }
                                }
                                else
                                {
                                    for (C_n[j] = Math.Max(Num_Velocity[j] - 2, 0);
C_n[j] <= Math.Min(Num_Velocity[j] + 2, VC_n); C_n[j]++)
                                    {
                                        t_f[j] = (int)Math.Ceiling(r[j] *
(double)(C_n[j]) / D + (1 - r[j]) * Math.Max(0, Math.Min((double)(C_n[j]) / D, t_safe) -
1));
                                        if (t_f[j] == 0)
                                        {
                                            aa[j] = 0;
                                        }
                                        else
```

```
                                        {
                                            aa[j] = 1;
                                        }
                                        if (C_n[j] % D == 0)
                                        {
                                            if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] + 1) * t_f[j] * D))
                                            {
                                                C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                                break;
                                            }
                                            else
                                            {
                                                C_n1[j] = C_n[j];
                                                continue;
                                            }
                                        }
                                        else
                                        {
                                            if ((t_f[j] + 1) * C_n[j] >
(int)(Distance[j] + 0.5 * (t_f[j] - 1) * t_f[j] * D + aa[j] * (2 * t_f[j] - 1)))
                                            {
                                                C_n1[j] = Math.Max(C_n[j] - 1, 0);
                                                break;
                                            }
                                            else
                                            {
                                                C_n1[j] = C_n[j];
                                                continue;
                                            }
                                        }
                                    }
                                }
                            }
                        }

                        #endregion

                        #region
                        int[] ad = new int[5000];
                        if (tt > green - 4 && tt < green + 5)
                        {
                            int cc = 0;
                            for (int j = Number - 1; j > 0; j--)
                            {
                                if (stgo[j] == 2)
                                    cc++;
                                if (Num_Location[j] > 4920 && Num_Location[j] < 4995 &&
(j == Number - 1 || (stgo[j] == 2 && cc == 0)))
                                {
                                    {
                                  double gap = (double)Distance[j] / Num_Velocity[j];
                                        int stbar = 4995 - Num_Location[j];
                                        ad[j] = pp.AccDec(gap, stgo[j], stbar,
Num_Velocity[j]);
                                    }
                                }
```

```
                        }
                    }

                    #endregion
                    #region
                    int[] Num_Velocity_Last = new int[N];
                    for (int j = 0; j < Number; j++)
                    {
                        switch (ad[j])
                        {
                            case 0:
                                Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Num_Velocity[j] + 1,
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                break;
                            case 1:
                                Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Num_Velocity[j] + 1,
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                break;
                            case 2:
                                Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Math.Max(Num_Velocity[j] - 1, 0),
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                break;
                            case 3:
                                Num_Velocity_Last[j] = Num_Velocity[j];
                                break;
                            case 4:
                                Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Math.Max(Num_Velocity[j] - 1, 0),
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                break;
                            case 5:
                                Num_Velocity_Last[j] =
Math.Min(Desired_speed[Num_Location[j]], Math.Min(Math.Max(Num_Velocity[j] - 2, 0),
Math.Max(Math.Max(0, Num_Velocity[j] - D), C_n1[j])));
                                break;
                        }
                    }
                    #endregion
                    #region
                    for (int j = 0; j < Number; j++)
                    {
                        double sj = random.Next(0, 999);
                        sj /= 1000;
                        if (((signal_control == 1 || signal_control == 2) &&
Num_Location[j] >= 4995) || kkk[Number - 1] == 1)
                        {
                            continue;
                        }
                        if (sj < 0.16)
                        {
                            Num_Velocity_Last[j] = Math.Max(0,
Math.Max(Num_Velocity[j] - D, Num_Velocity_Last[j] - 1));
                        }
                    }
                    for (int j = 0; j < Number; j++)
```

```
{
    Num_Velocity[j] = Num_Velocity_Last[j];
}
#endregion
for (int j = Number - 1; j >= 0; j--)
{
    if (j < Number - 1)
    {
        if (Num_Velocity[j] - Num_Velocity[j + 1] > Distance[j])
        {
            Num_Velocity[j] = Num_Velocity[j + 1] + Distance[j];
        }
    }
    else
    {
        if (Num_Velocity[j] > Distance[j])
        {
            if (j == Number - 1 && Distance[j] < 0)
                Num_Velocity[j] = 0;
            else
                Num_Velocity[j] = Distance[j];
        }
    }
}

for (int j = 0; j < Number; j++)
{
    if (Num_Velocity[j] < 0)
    {
        error++;
    }
}

#region
k = 0;
kii = ki;
if (t >= TT1 - 2)
{
    for (int j = 0; j < Number; j++)
    {
        Velocity_2[j] = Velocity_1[j];
    }
}
for (int j = 0; j < Number; j++)
{
    Num_Location_1[j] = Num_Location[j] + Num_Velocity[j];
    Velocity_1[j] = Num_Velocity[j];
    if (j <= Number - 1 && j >= ki)
    {
        if (t > TT1 && Velocity_2[j - ki] - Num_Velocity[j] > 2)
        {
            Slam_Brake1[t - TT1 - 1]++;
                        if (t_l22 < 5000)
            {
                t_l2[0, t_l22] = tt;
                t_l2[1, t_l22] = Num_Location[j];
                if (Num_Location[j] < 4900)
                    error++;
```

```
                                    t_l2[2, t_l22] = alpha;
                                    t_l2[3, t_l11] = Desired_speed[Num_Location[j]];
                                    t_l22++;
                                }
                            }
                        }
                        if (Num_Location_1[j] >= Road_Length)
                        {
                            k++;
                            if (t >= TT1)
                            {
                                Flux[(int)Math.Round(100 * (alpha - 0.01))]++;
                            }
                        }

                    }
                    if (t > TT1)
                    {
                        Slam_Brake1[t - TT1 - 1] /= Number;
                    }
                    for (int j = 0; j < Number - k; j++)
                    {
                        Desired_speed_1[Num_Location_1[j]] =
Desired_speed[Num_Location[j]];
                    }
                    for (int j = 0; j < Number; j++)
                    {
                        Num_Location[j] = 0;
                    }
                    for (int j = 0; j < Number - k; j++)
                    {
                        Num_Location[j] = Num_Location_1[j];
                    }
                    for (int j = 0; j < Road_Length; j++)
                    {
                        Location[j] = 0;
                        Velocity[j] = 0;
                        Desired_speed[j] = 0;
                    }
                    for (int j = 0; j < Road_Length; j++)
                    {
                        Desired_speed[j] = Desired_speed_1[j];
                    }
                    for (int j = 0; j < Number - k; j++)
                    {
                        Location[Num_Location[j]] = 1;
                        Velocity[Num_Location[j]] = Num_Velocity[j];
                    }

                    #endregion

                    if (Number > 5)
                    {
                        for (int j = 0; j < Number - Math.Max(1, k); j++)
                        {
                            if (Num_Location[j] >= Num_Location[j + 1] &&
Num_Location[j + 1] != 0)
                            {
```

```
                                error++;
                            }
                        }
                    }
                }

                #region
                for (int j = 0; j < TT; j++)
                {
                    DSZ1[(int)Math.Round(100 * (alpha - 0.01))] += DS1[j];
                    DSZ2[(int)Math.Round(100 * (alpha - 0.01))] += DS2[j];
                    Slam_Brake[(int)Math.Round(100 * (alpha - 0.01))] +=
Slam_Brake1[j];
                }
                DSZ1[(int)Math.Round(100 * (alpha - 0.01))] /= TT;
                DSZ2[(int)Math.Round(100 * (alpha - 0.01))] /= TT;
                Slam_Brake[(int)Math.Round(100 * (alpha - 0.01))] /= TT;

                #endregion
            }
            for (int j = 0; j < 100; j++)
            {
                DSZ11[j] += DSZ1[j];
                DSZ22[j] += DSZ2[j];
                Flux11[j] += Flux[j];
                Slam_Brake11[j] += Slam_Brake[j];
                if (DSZ11[j] == 0 || DSZ22[j] == 0)
                {
                    error++;
                }
            }
        }
        for (int j = 0; j < 100; j++)
        {
            DSZ11[j] /= CC;
            DSZ22[j] /= CC;
            Flux11[j] /= CC;
            Slam_Brake11[j] /= CC;
        }
        // dangerous situation
        FileStream dsz1 = File.Create("E:\\DSZ1.txt");
        FileStream dsz2 = File.Create("E:\\DSZ2.txt");
        StreamWriter swdsz1 = new StreamWriter((System.IO.Stream)dsz1);
        StreamWriter swdsz2 = new StreamWriter((System.IO.Stream)dsz2);
        for (int j = 0; j < 100; j++)
        {
            swdsz1.Write("{0} ", DSZ11[j]);
            swdsz2.Write("{0} ", DSZ22[j]);
        }
        swdsz1.Close();
        swdsz2.Close();
        // flux
        FileStream flow = File.Create("E:\\FLUX.txt");
        StreamWriter swflow = new StreamWriter((System.IO.Stream)flow);
        for (int j = 0; j < 100; j++)
        {
            swflow.Write("{0} ", Flux11[j]);
        }
```

```
        swflow.Close();
        FileStream brake = File.Create("E:\\Brake1.txt");
        StreamWriter swbrake = new StreamWriter((System.IO.Stream)brake);
        for (int j = 0; j < 100; j++)
        {
            swbrake.Write("{0} ", Slam_Brake11[j]);
        }
        swbrake.Close();
        FileStream st1 = File.Create("E:\\tl1.txt");
        StreamWriter stw1 = new StreamWriter((System.IO.Stream)st1);
        FileStream st2 = File.Create("E:\\tl2.txt");
        StreamWriter stw2 = new StreamWriter((System.IO.Stream)st2);
        for (int jj = 0; jj < 4; jj++)
        {
            for (int j = 0; j < 5000; j++)
            {
                stw1.Write("{0} ", t_l1[jj, j]);
                stw2.Write("{0} ", t_l2[jj, j]);
            }
            stw1.Write("\n");
            stw2.Write("\n");
        }
        stw1.Close();
        stw2.Close();

        FileStream stgo_Error = File.Create("E:\\stgo_error.txt");
        StreamWriter sger = new StreamWriter((System.IO.Stream)stgo_Error);

        for (int j = 0; j < stgo_para; j++)
        {
            sger.Write("{0} ", stgo_error[j]);
        }
        sger.Close();

                    #endregion

        Console.WriteLine("OVER");
        Console.ReadLine();
    }
    public int StopGo(int di, int sp, bool lf)
    {

        #region
        Random aa = new Random();
        if (di > 4941)
        {
            double cc = aa.Next(0, 999) / 1000;
            if (cc < 0.191)
                return 2;
            else
                return 1;
        }
        else
            if (di < 4924)
            {

                double cc = aa.Next(0, 999) / 1000;
                if (cc < 0.922)
```

```
                return 2;
            else
                return 1;
        }
        else
            if (sp < 15)
            {
                double cc = aa.Next(0, 999) / 1000;
                if (cc < 0.604)
                    return 2;
                else
                    return 1;
            }
            else
                if (lf)
                {
                    double cc = aa.Next(0, 999) / 1000;
                    if (cc < 0.488)
                        return 2;
                    else
                        return 1;
                }
                else
                {
                    double cc = aa.Next(0, 999) / 1000;
                    if (cc < 59.6)
                        return 2;//2stop
                    else
                        return 1;
                }
    #endregion
}

public int AccDec(double gap, int stopgo, int Dtostopbar, int speed)
{
    Random pa = new Random();
    if (stopgo == 1)
    {
        if (gap <= 3)
        {
            double aa = pa.Next(0, 1000);
            aa /= 1000;
            if (aa < 0.7686)
                return 2;
            else
                return 3;
        }
        else if (gap >= 3.6)
        {
            double aa = pa.Next(0, 1000);
            aa /= 1000;
            if (aa < 0.07)
                return 1;
            else
                return 3;
        }
        else
            return 3;
```

```
            }
            else
            {
                int d1, d2;
                d1 = speed * (speed - 1) / 2;
                if (speed % 2 == 0)
                    d2 = speed * (speed - 2) / 4;
                else
                    d2 = (speed - 1) * (speed - 1) / 4;
                if (Dtostopbar > d1)
                {
                    if (Dtostopbar - speed > d1)
                        return 0;
                    else if (Dtostopbar - speed - 1 > (speed - 1) * (speed - 2) / 2)
                    {
                        return 4;
                    }
                    else
                    {
                        return 5;
                    }
                }
                else if (Dtostopbar > d2)
                    return 5;
                else
                    return 5;
            }
        }
        public double ST_GO_LogitRegression(bool lead, int spe, int dis)
        {
            int x1 = 0, x2 = 0, x3 = 0, x4 = 0, x5 = 0, x6 = 0;
            if (lead)
                x1 = 1;

            if (spe > 14 && spe < 16)
                x2 = 1;
            else
                if (spe < 20)
                    x3 = 1;
            if (dis > 57 && dis < 79)
                x4 = 1;
            else
                if (dis >= 79 && dis < 88)
                    x5 = 1;
                else
                    if (dis >= 88 && dis < 98)
                        x6 = 1;
            double y = Math.Exp(-0.1945 + 0.9350 * x1 + 1.4994 * x2 + 3.2820 * x3 -
2.4108 * x4 - 4.5557 * x5 - 5.2498 * x6) /
                (1 + Math.Exp(-0.1945 + 0.9350 * x1 + 1.4994 * x2 + 3.2820 * x3 - 2.4108
* x4 - 4.5557 * x5 - 5.2498 * x6));
            return y;
        }
        public double AverageRandom(double min, double max)
        {
            double aa = random.Next(0, 1001);
            aa = aa / 1000;
            return min + aa * (max - min);
```

```
        }
        public double Normal(double x, double mu, double sigma)
        {
            return 1.0 / Math.Sqrt(2 * Math.PI * sigma) * Math.Exp(-1 * (x - mu) * (x -
mu) / (2 * sigma * sigma));
        }
        public int NormalRandom()
        {
            double x;
            double y;
            double dScope;
            do
            {
                x = AverageRandom(34, 66);
                dScope = Normal(x, 50, 5);
                y = AverageRandom(0, 0.14);
            }
            while (y >= dScope);
            x = (int)Math.Round(x * 0.44704 / 1.5, 0);
            return (int)x;
        }
    }
}
```

APPENDIX E SIMULATION RESULTS-PROBABILITIES OF RISKY SITUATIONS

P-BRAKE	30,5	40,5	50,5	60,5	50,2	50,10
Typical	0.000411	0.00052	0.000644	0.000951	9.32E-05	0.000676
Flashing Green	0.00044	0.000485	0.000647	0.001507	0.000139	0.000631
Pavement Marking	0.000382	0.000446	0.000522	0.000792	5.66E-05	0.000582
pavement marking and an auxiliary indication	8.67E-05	0.000126	0.000144	0.000779	1.9E-05	0.000193
P-RS1	**30,5**	**40,5**	**50,5**	**60,5**	**50,2**	**50,10**
Typical	0.000208	0.000289	0.00044	0.000426	6.44E-05	0.000431
Flashing Green	0.000228	0.000265	0.000447	0.00081	0.000107	0.000421
Pavement Marking	0.000189	0.000229	0.000359	0.000371	3.84E-05	0.000382
pavement marking and an auxiliary indication	3.93E-05	6.2E-05	8.28E-05	0.000345	9.72E-06	0.00011
P-RS2	**30,5**	**40,5**	**50,5**	**60,5**	**50,2**	**50,10**
Typical	1.9E-05	3E-05	2.88E-05	4.12E-05	3.08E-06	2.75E-05
Flashing Green	1.84E-05	2.88E-05	2.99E-05	4.9E-05	2.17E-06	2.54E-05
Pavement Marking	6.51E-06	9.41E-06	8.08E-06	1.22E-05	7.49E-07	9.45E-06
pavement marking and an auxiliary indication	1.07E-06	5.38E-06	5.6E-06	1.01E-05	7.88E-07	5.02E-06
P-RLR	**30,5**	**40,5**	**50,5**	**60,5**	**50,2**	**50,10**
Typical	0.0037	0.0056	0.0166	0.0052	0.0323	0.0134
Flashing Green	0.0037	0.0084	0.0205	0.0085	0.0300	0.0138
Pavement Marking	0.2724	0.2035	0.1072	0.0188	0.0114	0.2183
pavement marking and an auxiliary indication	0	0.0006	0.0003	0.0001	0.0001	0.002

LIST OF REFERENCES

Clarridgea A. and Salomaab K. (2010). Analysis of a cellular automaton model for car traffic with a slow-to-stop rule. Theoretical Computer Science, 411(38-39), 3507-3515. doi: 10.1016/j.tcs.2010.05.027

Federal Highway Administration. How red-light running is defined and how crash figures are determined. Retrieved Aug.14, 2014, from http://safety.fhwa.dot.gov/intersection/redlight/howto/

Boccara N., Fuks H. and Zeng, Q. (1997). Car accidents and number of stopped cars due to road blockage on a one-lane highway. J.Phys.A:Math.Gen, 30, 3329-3332.

Messer C., Sunkari, S., Charara, H. and Parker, R. (2003). Development of advance warning systems for end-of-green phase at high speed traffic signals: Texas Transportation Institute.

Chang M., Messer C. and Santiago A. (1985). Timing traffic singal change intervals based on driver behavior. Transportation Research Record: Journal of the Transportation Research Board, 20-30.

Chiou Y. and Chang C. (2010). Driver responses to green and red vehicular signal countdown displays: Safety and efficiency aspects. Accident Analysis and Prevention, 42(4), 1057-1065. doi: 10.1016/j.aap.2009.12.013

Retting R., Williams A., Farmer C. et al. Evaluation of red light camera enforcement in Oxnard, California, Accident Analysis & Prevention, Volume 31, Issue 3, May 1999, P169-174

Liu C. and Herman R. (1996). A review of the yellow interval dilemma. Transportation Research Part A: Policy & Practice, 30, 333-348.

ITE Technical Committee. (1989). Determing vehicle change intervals: a propsed recommended practice ITE Journal (Vol. 57). Washington D.C.: Institute of Transportation Engineers.

Huiwitz D. and Knodler M. (2007). Static and dynamic evaluation of the driver speed perception and selection process. Paper presented at the Proceedings of the 4th International Driving Symposium on Human Factors in Driver Assessment, Training & Vehicle Design, Stevenson, Washington, USA.

Gazis D., Herman R. and Maradudin A. (1959). The porblem of the amber signal light in traffic flow.

Ding Y., Wu Y., Abdel-Aty M. et al. (2014). Studying rear-end crash risk under countermeasures of amber dilemma at signalized intersections based on cellular automaton model. Manuscript under review in Accident Analysis & Prevention.

Google map. Retrieved Aug.01, 2014, from https://www.google.com/maps/@28.56486,-81.238201,14z?hl=en

Elmitiny N., Yan X., Radwan E. et al. (2010). Classification analysis of Driver's Stop/go Decision and Red-light Running Violation. Accident Analysis & Prevention, 42(1), 101-111. doi: 10.1016/j.aap.2009.07.007

Porter B. and Kelli E. (2000). Predicting red-light running behavior: a traffic safety study in three urban settings. Journal of Safety Research, 31, 1-8.

Factor R., Prashker J. and Mahalel D. (2012). The flashing green light paradox. Transportation Research Part F: Traffic Psychology and Behaviour, 15(3), 279-288. doi: 10.1016/j.trf.2012.01.003

Hurwitz D. (2009). Application of driver behavior and comprehension to dilemma zone definition and evaluation. (dissertation), University of Massachusetts

Bonneson J., Middleton D., Zimmerman K., Charara H. and Abbas M. (2002). Intelligent detection-control system for rural signalized intersections: Texas Transportation Institute.

Jia B., Gao Z., Li K. et al. (2007). Models and simulations of traffic system based on the theory of cellular automaton. Beijing.

Jiang R., Wang X., Jia B. et al. (2003). Dangerous situations within the framework of the Nagel-Schreckenberg model. JOURNAL OF PHYSICS A-MATHEMATICAL AND GENERAL, 36(17), 4763-4769.

Rui J., Bin J., Wang, X. and Wu Q. (2004). Dangerous situations in the velocity effect model. Journal of Physics A: Mathematical and General, 37(22), 5777-5787. doi: 10.1088/0305-4470/37/22/005

Knodler M., Noyce D., Kacir K. et al. (2001). Driver understanding of the green ball and flashing yellow arrow permintted indications: a driving simulator Experiment. ITE Journal.

Lum K. and Halim H. (2006). A before-and-after study on green signal countdown device installation. Transportation Research Part F: Traffic Psychology and Behaviour, 9(1), 29-41. doi: 10.1016/j.trf.2005.08.007

Köll H., Bader M. and Axhausen K. (2004). Driver behaviour during flashing green before amber: a comparative study. Accident Analysis & Prevention, 36(2), 273-280. doi: 10.1016/s0001-4575(03)00005-8

Lidia K.and Eby D. (1998). Exploring rear-end roadway crashes from the driver's perspective: The University of Michigan.

Liu Y. , Chang G., Tao R. et al. (2007). Empirical observations of dynamic dilemma zones at signalized intersections. Transportation Research Record: Journal of the Transportation Research Board, 122-133.

Lum K. and Wong, Y. (2003). Impacts of red light camera on violation characteristics. Journal of Transportation Engineering. doi: 10.1061//ASCE/0733-947X/2003/129:6/648

Mahadlel D. , Zaidel D. and Klein T. (1985). Drivers decision process on termination of the green light. Accident Analysis and Prevention, 17(5), 373-380.

Mahalel D. and Zaidel D. (1985). Safety evaluation of a flashing green light in a traffic signal. Traffic Engineering & control, 26(2), 79-81.

Manual of Uniform Traffic Control Devices. (2009). Washington, DC: Federal Highway Administration.

Manual on uniform traffic control devices: for streets and highways. (2009) (pp. 485): Federal Highway Administration.

Abdel-Aty M. and Abdelwahab H. (2004). Modeling rear-end collisions including the role of driver's visibility and light truck vehicles using a nested logit structure. Accident Analysis & Prevention, 36(3), 447-456. doi: 10.1016/s0001-4575(03)00040-x

Nagel K. and Michael S. (1992). A cellular automaton model for freeway traffic. journal de physique, 2, 2221-2229.

Newton C., Mussa R. Sadalla E. et al. (1997). Evaluation of an alternative traffic light change anticipation system. Accident Analysis and Prevention, 29(2), 201-209.

Papaioannou, P. (2007). Driver behaviour, dilemma zone and safety effects at urban signalized intersections in Greece. Accident Analysis & Prevention, 39(1), 147-158. doi: 10.1016/j.aap.2006.06.014

Pline, J. (1999). Traffic Engineering Handbook fifth Edition: Institute of Transportation Engineering.

Red light running. Retrieved Aug. 10, 2014, from http://www.iihs.org/iihs/topics/t/red-light-running/topicoverview

Jiang R. and Wu Q. (2006). A stopped time dependent randomization cellular automata model for traffic flow controlled by traffic light. Physica A: Statistical Mechanics and its Applications, 364(15), 493-496. doi: 10.1016/j.physa.2005.10.038

Kikuchi S. and Riegner J. (1992). Methodology to analyze driver decision environment during signal change: intervals application of fuzzy set theory. Transportation Research Board, 49-57.

Sunkai S., Messer C. and Charara H. (2005). Performance of advance warning for end of green system for high-speed signalized intersections. Transportation Research Record: Journal of the Transportation Research Board, 1925, 176-184.

Maerivoet S. and Moor B. (2005). Cellular auomata models of road traffic. Physics Reports. doi: 10.1016/j.physrep.2005.08.005

Andrew T., Wei L. and Luis L. (2006). A probabilistic approach to control dilemma zone occurrence at signalized intersections Presented at 85th Annual Meeting of the Transportation Research Board.

Gates T., Noyce D. and Luis L. (2006). Analysis of dilemma zone driver behavior at signalized intersections. Transportation Research Record: Journal of the Transportation Research Board, 2030, 29-39.

Traffic safety facts 2012. (2012). Washington, DC: National Highway Traffic Safety Administration.

Stephen W. (1983). Statistical mechanics of cellular automata. Reviews of Modern Physics, 55(3), 601-644.

Wortman R. and Matthias J. (1983). Evaluation of Driver Behavior at Singalized Intersections. Transportation Research Record: Journal of the Transportation Research Board, 10-20.

Yan X., Radwan E. and Abdel-Aty M. (2005). Characteristics of rear-end accidents at signalized intersections using multiple logistic regression model. Accid Anal Prev, 37(6), 983-995. doi: 10.1016/j.aap.2005.05.001

Yan X., Radwan E., Guo D. et al. (2009). Impact of "Signal Ahead" pavement marking on driver behavior at signalized intersections. Transportation Research Part F: Traffic Psychology and Behaviour, 12(1), 50-67. doi: 10.1016/j.trf.2008.07.002

Yan X., Radwan E. and Guo D. (2007). Effect of a pavement marking countermeasure on improving signalized intersection safety. ITE Journal, 77(8), 30-39.

Ni Y. and Li K. (2014). Estimating rear-end accident probabilities at signalized intersections: a comparison study of intersections with and without green signal countdown devices. Traffic Injure Prevention, 15(6), 583-590. doi: 10.1080/15389588.2013.845752

Han Y. and Ko S.. (2012). Analysis of a cellular automaton model for car traffic with a junction. Theoretical Computer Science, 450(7), 54-67. doi: 10.1016/j.tcs.2012.04.027

Zegeer C. and Deen R.(1978). Green-extension systems at high-speed intersections. ITE Journal, 48, 19-24.