8-2015

A Compact Driving Simulator to Support Research and Training Needs - Hardware, Software, and Assessment

QIMIN YAO
Clemson University, yaoqimin@gmail.com

A COMPACT DRIVING SIMULATOR TO SUPPORT RESEARCH AND TRAINING NEEDS – HARDWARE, SOFTWARE, AND ASSESSMENT

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master Science
in Mechanical Engineering

By
Qimin Yao
December 2012

ABSTRACT

In 2009, there were 10.8 million automotive crashes in the United States. When reviewing these incidents, young drivers were the over-represented group who were involved in these vehicular crashes. One reason that these young drivers have such high crash rates is that they often do not possess extensive driving knowledge and typically lack driving experience. Over the past three decades, researchers have investigated the driver training process which includes the classroom and behind the wheel to help address teaching the deficiencies in the driver education program. To complement the current driver education process, automotive simulators can be utilized to teach skills and strategies. Driving simulators offer multiple advantages as student drivers are accustomed to playing video games, it is entirely safe to practice driving in a virtual environment, and many different types of driving tracks and custom scenarios can be implemented. In this thesis, an innovative automotive simulator will be created and tested to evaluate its effectiveness for teaching novice drivers fundamental motor vehicle skills.

The personal computer-based portable Clemson Automotive Training System (CATS) was developed to teach young drivers the necessary driving knowledge and then allow them to practice driving a vehicle on a virtual track. The students became familiarized with typical road situations and simultaneously enhanced their driving abilities when facing unexpected scenarios. In CATS, a track and four different scenarios (e.g., “Control Sign”, “Lane Selection”, “Braking”, and “Obstacle Avoidance”) were created to test different driving aspects. The driving rating and feedback was displayed on the computer screen immediately after the student completed the driving test. As part of the experimental study, 50 participants were invited to complete the virtual driving training process. The driving test contained three parts: a pre-test questionnaire, four driving scenarios, and a post-test questionnaire. The laboratory results show that the novice drivers demonstrated a significantly higher potential in acquiring basic driving knowledge and improving their driving skills after practicing on a CATS simulator than more experienced drivers. There was an overall driving improvement of 28% after the students finished the CATS. The simulator has successfully demonstrated its ability to enhance both driving knowledge and driving skills through the training of novice drivers.

Further development of CATS is necessary to refine and improve its ability to educate and train novice drivers. A more realistic track should be constructed with high-resolution objects such as a school zone and buildings along the roadway, and more scenarios designed to expore more advanced driving skills. In addition, an additional force feedback function should be added to enhance the driving realism. Finally, vehicular traffic should be added to the simulator environment that will interact with each student's vehicle to offer a more realistic scenario.

DEDICATION

This thesis is dedicated to my mother and father, Xiaoping Yuan and Ronghua Yao, for their support and understanding.

I would like to thank my wife, Xiaowen Zhu, who is sharing my life. For it can be stated, “Life without you is like a flower without sunshine.”

ACKNOWLEDGMENTS

I am proud to be a student at Clemson University and happy to pursue my masters degree in Mechanical Engineering. I want to thank Professor John Wagner for not only being a great teacher but also for being an incredible director. I would also like to thank the members of the Clemson University Automotive Safety Research Institute (CU ASRI) including Dr. Kim Alexander, Dr. Philip Pidgeon, Ms. Eleanor Walters, Mr. Christopher Ball, and Ms. Catherine Murphree for providing financial aid and granting me the opportunity to work on this project. In addition, I wish to thank Dr. Gregory Mocko to become my committee member and offer suggestions on my thesis. Moreover, I would like to thank Mr. Lance Clark and Mr. Christopher Guirl for their assistance on the project. Also, I would like to thank to Karl Stefan and Ryan Mattfeld for their assistance during the thesis process. Finally, I want to express my appreciation to the Department of Mechanical Engineering staff, including Ms. Gwen Dockins, Ms. Tameka Boyce, and Ms. Kathryn Poole for their help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

List of Figures (Continued)

List of Figures (Continued)

NOMENCLATURE LIST

A	Deceleration (m/s2)
D	Braking distance (m)
DR	Driver rating
DS	Driver score
DY	Driver years
F	Obey traffic rule logical variable
$\bar{K}_{ij}$	Driving factor metric score
L	Distance between vehicle centerline and middle of road (m)
N1	Number of times vehicle traveled off road
N2	Number of times vehicle ventured across double yellow line
N3	Number of times vehicle traveled faster than posted speed limit
R	Reaction time (sec)
S	Module score
T	Braking time (sec)
V	Vehicle speed (m/s)
Vp	Peak velocity (m/s)
V_{peak}	Peak velocity improvement between 1st lap and 2nd lap
V_{ave}	Average velocity improvement between 1st lap and 2nd lap
V_{ave1}	One time velocity during the 1st lap (m/s)
V_{ave2}	One time velocity during the 2nd lap (m/s)
V_{p1}	Peak velocity during the 1st lap, (m/s)
V_{p2}	Peak velocity during the 2nd lap, (m/s)

GREEK SYMBOLS

α_{ij}	Weighted score for each driving factor
β	Penalty factor
δ_1	One time steering angle during the 1st lap, (deg)
δ_2	One time steering angle during the 2nd lap, (deg)
δ_{p1}	The maximum steering angle during the 1st lap, (deg)
δ_{p2}	The maximum steering angle during the 2nd lap, (deg)
δ_{ave1}	Average steering angle on the 1st lap, (deg)
δ_{ave2}	Average steering angle on the 2nd lap, (deg)
δ_{ave}	Average steering angle improvement between 1st lap and 2nd lap
δ_{peak}	Peak steering angle improvement between 1st lap and 2nd lap

Nomenclature List (Continued)

SUBSCRIPT

i	Module number
j	Module driving assessment factor
n	Total number of modules
t	Total time of driving on the lap, (sec)
u	Difference between deceleration. A, and 0.8 (m/s2)
x	X-coordinate position (m)
y	Y-coordinate position (m)

CHAPTER ONE

INTRODUCTION

Young drivers are involved in a larger proportion of automobile crashes resulting in injuries or death when compared to other driver population groups. In 2010, 1,963 young people whose ages ranged between 15 and 20 years old were killed and another 187,000 were injured in automobile crashes in the United States. The crash rate of young drivers has become an important societal issue, and many groups are working to reduce this statistic. Studies have shown that there are four major factors that contribute to the higher crash rates among young drivers when compared to other driver populations such as experienced drivers.

The first factor that causes an increase in crashes among young drivers is their lack of experience, and may not fully understand traffic signs. For example, novice drivers may not notice yield signs at intersections, and consequently, they may drive through the intersection without stopping, which is very dangerous. The second factor that causes higher accident rates among young drivers is that they do not see and react to hazards as quickly as experienced drivers because they may not be familiar with their vehicle, or they may not even notice the hazard. The third factor is that some young drivers over-estimate their driving skills. This over-estimation is created because novice drivers have only watched more experienced drivers, such as their parents and likely played racing video games at home. Being limited to this experience creates a simplified view of driving, thus making it seem easier than it is in real life. Due to the over-estimation of driving skills, when they meet unexpected events, they are not prepared, which leads accidents. The fourth factor leading to increased accidents is distractions. With new technology in society on a regular basis, young drivers can interact with mp3 players, cell phones, portable video games, and many others. These instruments can distract drivers and cause them to take their eyes off the road, leading to accidents.

Video games that simulate driving are quite popular among young people. Many of them like playing racing games by themselves or with their friends for entertainment. Among these racing games, “Grand Turismo Five” and “Needs For Speed” are popular. These racing games create a virtual driving experience that is very similar to actual driving and have some common merits. First, they let people feel like they are actually steering a sports car on the road, even though they are just holding a controller and looking at a monitor. Second, the video games are safe; there's no danger when sitting at home and virtually driving. It is far safer not only for the driver, but also for the people that would have been near the driver if they were driving a real car. Third, tracks and modules can be specifically designed to test different driving situations. For example, a driving game can simulate a city, where there are many vehicles around to depict traffic jams. Fourth, people can practice driving on the same track multiple times, until they are familiar enough with the roadway to master all the traffic rules. Fifth, data, such as peak velocity, average velocity, peak steering angle, and yaw angle, can be collected easily with a driving simulator. Therefore, a virtual driving simulator will be an effective tool for educating young drivers.

Several researchers support the idea of implementing virtual driving simulator technology to train novice drivers. Lee (2007) recommended allowing them to practice driving on a virtual driving simulator while they go through the General Drivers License program. Similarly, a report shows that driving simulators are effective tools for evaluating novice drivers' hazard anticipation, speed management, and ability to maintain their focus on the road (Chan *et al.*, 2010). Crundall *et al.* (2010) proved that commentary training improves responsiveness to hazards in a driving simulator. Norfleet *et al.* (2009) showed that driving simulators are effective at communicating automotive safety lessons when followed by driving exercises to practice and reinforce the educational concepts. Some large-scale virtual driving simulators have been introduced both in commercial development and education. The automobile manufacturer, Fiat, uses a virtual driving tool to assess the external visibility of new car models to compare different geometries. Groot *et al*. (2012) used a driving simulator to investigate whether concurrent bandwidth feedback improves how well young drivers are able to stay in their lane while driving. Some researchers, such as Kandhai *et al*. (2011), are working on simulating driving on mobile apps.

While a number of large-scale and mobile phone based driving simulators have been developed, they tend to have significant weaknesses. First, it is hard to move larger driving simulators from one place to another, which means that if a person wants to use the simulator, they must travel to the simulator’s location. In addition, since the cost of large simulators is high, it is not practical to create a high number of them for the general populous. On the other hand, virtual phone simulators are not effective due to the hardware limitations. To solve these problems, a personal computer based virtual driving simulator is an ideal tool to educate and train novice drivers since it is both portable and offers high performance.

1.1 Hardware of CATS

A personal computer based simulator called the Clemson Automotive Training System (CATS) has been created. Its purpose is to educate and train novice drivers to become familiar with general driving skills. The CATS system has many merits as discussed above, including hardware capable of supporting a high-performance driving simulation, offering drivers a close approximation to the feeling of actually driving. In addition, it is portable, so it can be moved to schools and other community settings. The computer hardware includes a Dell OptiPlex 980 desktop with an Intel i7 processor, a 1GB high-performance graphics card, and a 19-inch monitor equipped with a sound bar. It also includes a racing chair, which can be moved forward and backward to adjust the distance between the driver and pedals, a Logitech G27 steering wheel with 900 degrees of rotation, and a foot pedal assembly containing the throttle, brake, and clutch. The steering wheel, pedals, and a seat belt are mounted to the racing chair assembly. The CATS software was developed based on VDrift. VDrift is an open source software package that effectively simulates vehicle dynamics and allows for the development of other customized modules much easier than most commercial driving software. The track is fully customized and can be altered to fit various learning environments. The track was built in Bob's Track Builder, a piece of commercial software. In this software, the programmer can design the shape of the track, add mountains, grass, or hills, and they can set the width of the road to either two lanes width or four lanes width as needed. The programmer can also set the coefficient of friction for the road surface used. The programmer can create modules on the track to simulate things like intersections, animals, some traffic obstacles, and traffic signs. With these adjustments, the track can be uniquely designed specifically for novice drivers to use. In addition, the programmer can create many customized objects on the track through the use of Blender, a piece of software that can create three-dimensional objects. The programming language used in CATS is C++, which allows for highly efficient programs.

1.2 New Features Developed in CATS

Using VDrift as a starting point, several new features were added to CATS to accomplish the goal of educating and training novice drivers. The first feature introduced was the ability to use an instantaneous feedback system to inform drivers about their performance. While drivers are using the simulator, the system continiuially displays information on the screen including messages and images whether the driver correctly handled a situation. Some of the scenarios use this feedback information to give the driver notification if there is an obstacle ahead on the track. The second feature added to CATS was a system that gives the driver a series of scores based on their performance. The scoring system has three parts. First, while people are driving, the score displays on the top left of the screen. It gives the driver their instant feedback score. Second, a scoring list is created when the driver opens the pause menu which allows them to check their score for each scenario. They also can view details for each item by clicking on a "Detail" button to the right of the scenario, which causes more specific information to display including their scores for each section of each scenario. Third, a background scoring system keeps more detailed information saved to an Excel file which becomes available after the driver has finished the simulation. The driver can check his or her score and data to identify where they can improve.

The third feature added to CATS was the addition of four scenarios that were implemented on the track. Each scenario has been designed to check one aspect of a novice driver's knowledge, and lead them about a specific scenario – stop sign, lane selection, panic braking, and animal avoidance. The Stop Sign Scenario detects if the driver knows the meaning of a stop sign and if they are capable of stopping their vehicle before the stop sign. The Lane Selection Scenario, in which one lane on the road is blocked, as if it were under construction detecting if the driver is able to change to the other lane on the road and slow down their vehicle to avoid the blocked lane. The Panic Braking Scenario requires the driver to speed up the vehicle to a high speed, and, after the velocity has reached a certain speed, it requires the driver to stop the vehicle immediately by displaying a panic stop sign image on the monitor. A detailed readout displays the deceleration of the vehicle, the stopping time, and the stopping distance on the bottom of the screen. The driver can see this information while driving, which allows them to better understand their vehicle's dynamics and their reaction time. The Animal Avoidance Scenario consists of an animal standing in the middle of the road, and the scenario tests whether the driver is able to react properly when they see the animal.

1.3 Safe Driving Program (SDP) and CATS

CATS can be applied in the Safe Driving Program (SDP) to help young drivers improve their driving knowledge and master important driving skills. The SDP is designed to give young drivers hands-on learning experiences aimed to train them to be safer drivers. The SDP contains two parts: the tent module and the in-vehicle modules (refer to Figure 1.1). The tent module teaches young drivers basic safe driving concepts and gives a preview of the road course that will be used for the in-vehicle modules. The in-vehicle modules focus on giving drivers both the mental and physical skills necessary for the young drivers to operate a vehicle in emergency situations. There is room for improvement in the current implementation of the SDP. First, young drivers don’t learn sufficient information by taking lectures in the tent module. They would learn the information better by seeing convincing data and more interesting animations. Second, when young drivers begin the in-vehicle modules, they may spend the first module or two simply trying to learn how to effectively operate their vehicle rather than learning the lessons presented. Third, there are only four modules in the program, which are not enough to simulate the majority of emergency driving scenarios. Thus, CATS would be an effective supplement for SDP. CATS can be applied between the tent module and the in-vehicle modules of SDP. The simulator can offer more modules for young drivers to practice which provides more opportunities to improve driving reactions. The simulator provides a safe way for young drivers to preview the in-vehicle modules and an easy method to add more modules to the SDP with minimal expenses. Moreover, CATS can simulate the dynamics of real vehicles, taking advantage of an effective physics engine to ensure a satisfactory driving experience (refer to Figure 1.2). CATS also provides quick and effective feedback, allowing young drivers to evaluate their strengths and weaknesses before they take the in-vehicle modules. In this way, CATS can increase the effectiveness of the SDP.

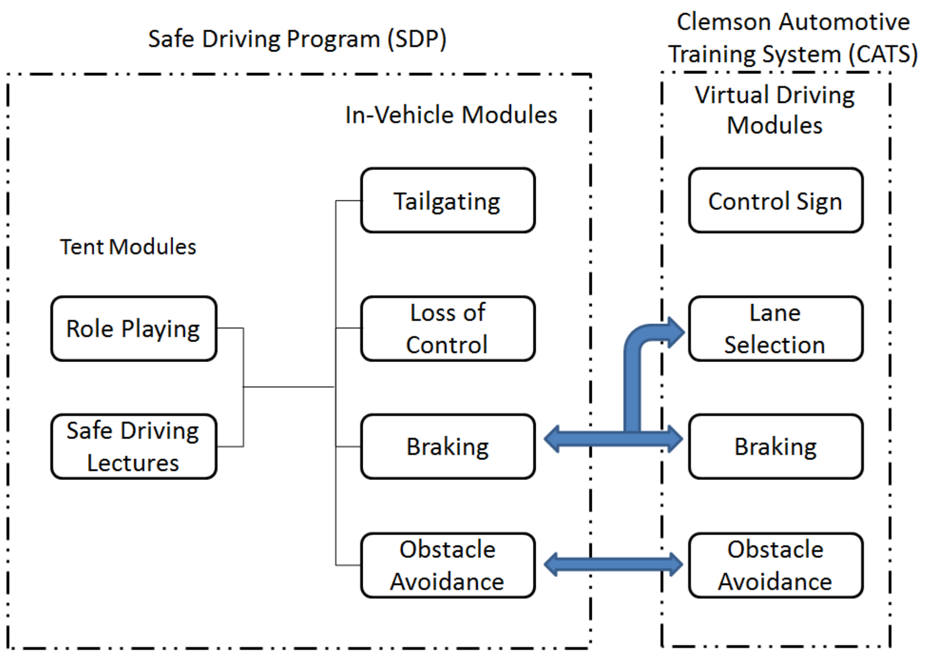


Figure 1.1 Relationships between SDP and CATS

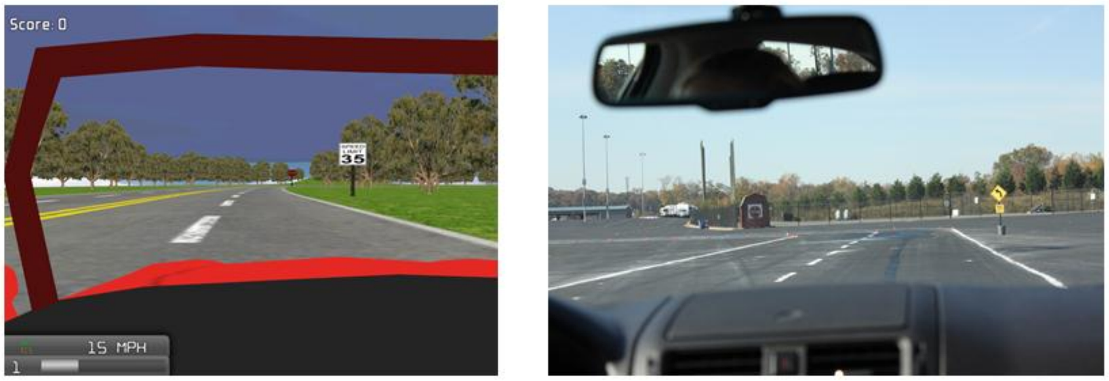


Figure 1.2 Comparison in-vehicle driving view between SDP and CATS

1.4 Thesis Organization

This thesis is divided into four chapters. Chapter 1 introduces the concept of the portable Clemson Automotive Training System (CATS) and establishes its goals. Chapter 2 introduces the details about the software and hardware of the portable CATS. It also covers the results of its pilot study. Chapter 3 evaluates more drivers' performances and discusses the results of their driving tests. Chapter 4 gives some conclusions to the effectiveness and outcome of the CATS. It also presents the challenges and further development of the CATS. The ten Appendices present the Pre and Post test questionnaire in safe driving program (Appendix A), the human test subjects' results (Appendix B and C), demographic questionnaire (Appendix D), the scoring metrics (Appendix E), the participant user manual (Appendix F), the develop user manual (Appendix G), and sample codes for Clemson Automotive Training System (Appendix H).

CHAPTER TWO

A VIRTUAL DRIVING EDUCATION SIMULATION SYSTEM -- HARDWARE AND SOFTWARE WITH PILOT STUDY

Novice drivers are often ill-equipped to safely operate a vehicle due to their limited repertoire of skills and experiences. However, automotive simulation tools can be applied to better educate young drivers for a number of common driving scenarios. In this paper, the Clemson Automotive Training System (CATS) will be presented to educate and train novice drivers to safely operate four wheel passenger vehicles on paved roadways. A portable automotive simulator can be programmed to emulate a variety of high-crash rate scenarios and roadway geometries. Drivers receive instructions regarding proper driving techniques and behaviors with an opportunity to practice the given vehicle maneuver. An on-line evaluation methodology has been designed to analyze the drivers' capabilities at handling these roadway events. First, a pre-simulation questionnaire evaluates their basic understanding of everyday driving situations. Second, when driving, individuals are assessed on multiple factors including vehicle speed, yaw angle, lateral acceleration, steering angle, and proper lane adherence while navigating the implemented track. Third, individuals complete a post-simulation questionnaire. Finally, a driving score is displayed on the computer screen to indicate the driver's performance. An initial human subject pilot study of novice, young, and seasoned drivers has been conducted. Overall, the driver performance improved an average of 12.75% (novice), 5.67% (young), and 4.31% (seasoned) which offers an indication of the driving simulation system's merits.

2.1 Introduction

In the United States for the 2010 calendar year, 1,963 people between the ages of 20-25 were killed and another 187,000 were injured from motor vehicle crashes (NHTSA, 2010). Studies have shown that younger drivers are more likely to be injured or killed during crash events than other driver populations (Deery and Flides, 1999). Several factors contribute to the high rate of motor vehicle crashes among young drivers. First, young drivers generally have less ability to detect hazards quickly as they tend to overestimate their general driving proficiency (Deery, 1999). Second, graduated driver licensing programs (GDL) only yield limited driving experiences (Ferguson, 2003) for young motorists and these programs do not have global participation. To properly prepare novice drivers for common road safety scenarios, additional resources should be offered using supplementary technology (Lee, 2007). Third, tele-communications and entertainment systems can lead to driver distractions, so young drivers need to be equipped with a proper attitudinal skill set to understand that operating a cell phone while driving often results in a delayed reaction (Muttart *et al.*, 2008). To help address these driver issues, an automotive simulator can be applied in the training process with targeted instructions and scenarios. These computer systems can help young drivers enhance their ability to identify potential traffic hazards (Vidotto *et al.*, 2011), improve their awareness (Chan *et al*., 2010), and simultaneously analyze their driving performance with real time feedback (Schultheis *et al.*, 2006).

Multiple studies and training programs have been piloted to decrease the crash rate among young drivers. In the 1950s, most public high schools in the United States introduced driver education programs (Lonero, 2008). In 1996, the AAA Foundation introduced the GDL program now used in all states and the District of Columbia (NHTSA, 2004). In recent years, universities and automotive manufacturers use driving simulators for research and product development activities. Clemson University designed and applied automotive simulators to train and educate young drivers (Norfleet *et al*., 2011). The University of Nottingham used a driving simulator to test whether a commentary driving system can remind drivers to notice potential hazards (Crundall *et al.,* 2010). The Korea's Automotive Technical Institute built a large scale virtual driving simulator to mimic the real world and analyze driver performance (Kim *et al.,* 2007). The University of Wisconsin-Madison introduced a variable message sign system and used a virtual driving simulation system to test its effectiveness (Dutta *et al.*, 2004). FIAT applied a driving simulator to develop and test new cars (Ruspa *et al.,* 2007). The University of Stuttgart created a virtual driving simulator platform to test and measure the fuel consumption and CO2 emissions (Piegsa *et al.*, 2011). In general, these simulators are often too large to regularly move to different locations and may not target novice drivers' needs. The Clemson Automotive Training System (CATS) has been developed using a PC-based platform, which should increase its application in high school, driving school and community settings. Figure 2.1 displays the roadway view from a third person perspective on the CATS computer screen.

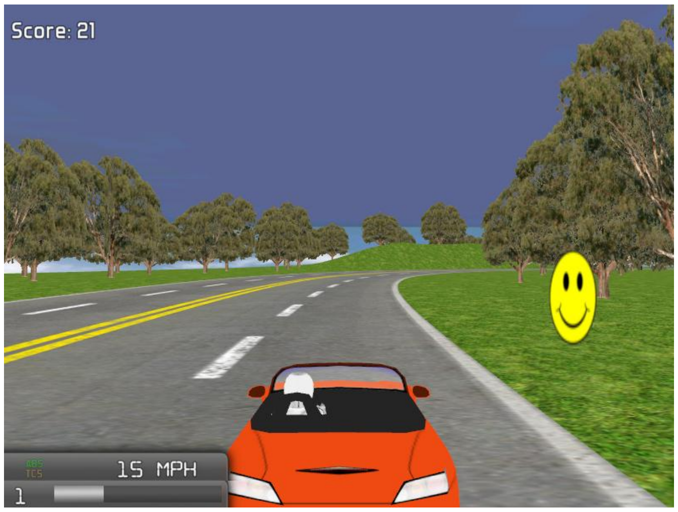


Figure 2.1: Virtual display of the user's vehicle along a straight roadway with universal feedback symbol on the right bottom of the screen indicating satisfactory driving performance

The CATS system was developed based on VDrift (Venjon, 2012), an open source software which, compared with most commercial driving software, was easier to develop. VDrift contains a good vehicle dynamic description. The tracks are fully customized and can be designed as the learning situation requires. Immense quantities of data can be collected within the virtual training environment and analyzed to evaluate a driver's performance. Further, CATS is a PC-based virtual driving simulator, which individuals can download and engage at home instead of going to a remote location provided they have the necessary driver interface hardware (refer to Section 2). To utilize the CATS, each participant will drive the simulation for two complete laps. For the first lap, students drive based on their own understanding of vehicle operation without any instructions. Four scenarios have been created that contain specific scenarios to evaluate the driver's performance. The drivers then view a video displayed on the screen which illustrates the correct manner to drive each scenario. The drivers complete a second lap on the same track. The computer system records both the pre-test and post-test questionnaire answers, as well as the driver's first and second simulation scores to assess their performance. The remainder of this paper is organized as follows. Section 2 introduces the system hardware and software for the CATS, as well as offers an overview of the four scenarios and their designed purposes. Section 3 reviews the analysis and assessment strategies. The instruction system directs drivers how to pass each scenario correctly. The scoring system records the drivers' behaviors and then rates their driving performance. Section 4 provides a pilot study with 12 human test subjects completing the automatic simulator training program. Finally, the conclusion is offered in Section 5. A complete nomenclature list is contained in the Appendix.

2.2 Hardware, Software Configurations and System Features

The virtual driving education simulator consists of three parts: computer, human-vehicle interface and real-time software. Figure 2.2 shows an image of the driving education simulator station. The computer hardware includes a Dell OptiPlex 980 desktop computer with Intel core i7 processor, a 19 inch monitor, and an attached speaker bar. A 1GB NVIDIA video card provides the graphics. The operating system is Ubuntu 10.10. A Logitech G27 steering wheel (900° range of rotation) with foot pedal assembly contains the throttle, brake pedal, and clutch. The steering wheel and pedals are mounted on a racing chair assembly with seat belt. The seat can move forward / backward to adjust the distance between the driver and pedals. The driving simulator was developed based on the VDrift, which is an open source driving software that uses the Vamos physics engine (Vamos, 2012). The track is created with Bob's Track Builder (BTB, 2012), a commercial building software, and Blender (Blender, 2012), a three dimensional building software. All the systems software features were written in the C++ language.

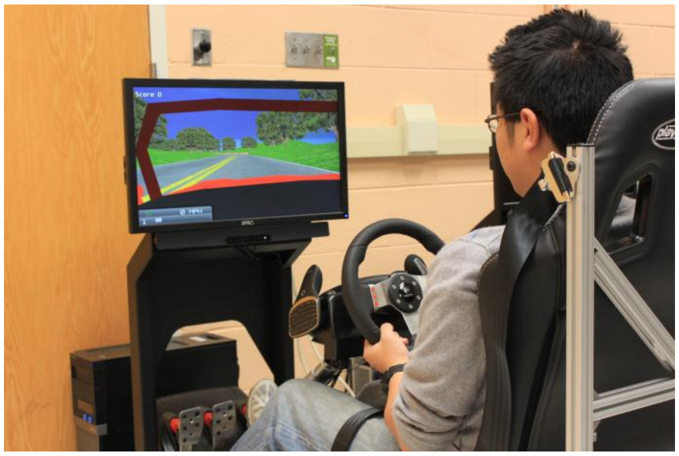


Figure 2.2: Portable Clemson Automotive Training System (CATS) with seat belt, standard human / vehicle interface, screen display, speaker and the host PC workstation

The novice drivers will receive training in several manners using the created software application. An instruction system shows interactive images and information during their driving time. A three part scoring system evaluates their driving skills; scores are displayed during and after the driving session. First, the simulator system will record the driver's data while they are driving and display a screen score on the top left. The score will update automatically after the vehicle has passed each scenario. Second, when the vehicle is approaching the next scenario, an image will be displayed to offer universal instruction about the roadway event. For example, when the vehicle is 30 meters close to the “lane selection” module, a road construction sign will be shown on the screen's right center directing the participant to the appropriate single lane which will allow them to pass the obstacle. Third, after the vehicle has driven through a scenario, the drivers' performances will display on the right bottom of the screen.

The scoring system collects and displays information during the pre-driving period to assess general knowledge and after the road driving portion. A set of ten questions are posed to gather attitudinal and knowledge data regarding driving behavior. Each driving scenario has certain requirements; the system will score the driver's performance over four scenarios, Si (i =1, 2, 3, 4), with each scenario worth a maximum of 100 points.

$$\mathbf{S}_i = \begin{Bmatrix} \sum_{j=7}^{3} \alpha_{ij} \overline{\mathbf{K}}_{ij}; & \text{for}\,(i=1,2,4) \\ \sum_{j=1}^{7} \alpha_{ij} \overline{\mathbf{K}}_{ij}; & \text{for}\,(i=3) \end{Bmatrix} \tag{2.1}$$

Where α_{ij} provides the weighting for each of the j evaluation factors, and $\overline{K_{ij}}$ contains the given metric's score. The system also counts the number of times the vehicle has traveled off the road, N1, the number of times the vehicle ventured across the double yellow line, N2, and the number of times the vehicle was traveling faster than the posted speed limit, N3. Specifically, each time the vehicle travels off the road, drives across the double yellow line, or exceeds the speed limit, a penalty of βk points will be deducted from the total score. Consequently, the driving score, $0 \leq \text{DS} \leq 100$, may be normalized as

$$\text{DS} = \frac{1}{n}\left(\sum_{i=1}^{4} S_i - \sum_{\kappa=1}^{3} \beta_\kappa N_\kappa \right) \tag{2.2}$$

where k denotes the penalty index (k=1, 2, 3) and n corresponds to the total number of scenarios. The relationship between the driving score and the driver rating, DR, can be expressed as

$$\text{DR} = \begin{cases} \text{Excellent;} & \text{if } 90 \leq \text{DS} \leq 100 \\ \text{Good;} & \text{if } 80 \leq \text{DS} < 90 \\ \text{Average;} & \text{if } 70 \leq \text{DS} < 80 \\ \text{Fair;} & \text{if } 60 \leq \text{DS} < 70 \\ \text{Dangerous;} & \text{if } \text{DS} < 60 \end{cases} \tag{2.3}$$

When the participant is driving the simulator, they can check the scoring menu for their current score. After a driver finishes the first loop, the total score will also display on the screen. At the same time, the table detailed score will be recorded in tabular function and stored on the hard drive. After finishing the simulator training session, the driver can view the complete assessment of their driving behavior.

2.3 Driving Scenarios

Four scenarios have been designed and implemented into the simulator to evaluate the driver's overall performance. Each scenario has its own special purpose. The first module (i=1) is an intersection with a stop sign that evaluates the drivers' behavior when approaching a four way stop. The second module (i=2), where one lane is closed, tests the drivers' ability to safely change lanes before entering a road construction zone. The third module (i=3) offers drivers an overview on vehicle dynamics for a high speed emergency stop. Finally, the last module (i=4) examines the driver's ability to avoid the sudden appearance of wildlife on the road. After these four scenarios, the driver should be more familiar with the rules of the road, gain an increased understanding regarding the vehicle dynamics involved in stopping a car within certain distances and its handling, and have a better appreciation for roadway hazards.

2.3.1 “Traffic Control Device” Module

The first module is designed to analyze drivers' performance, to increase novice drivers' familiarity with traffic signs, and to instruct them to react properly to each roadway sign. For example, stop signs (refer to Figure 2.3) are often not obeyed with a complete vehicle stop before the sign. Similarly, when traveling around a curve, the driver should turn the steering wheel and accelerate smoothly while keeping the car on track. To determine whether the driver successfully passes this scenario, three factors have been recorded: Vehicle speed when accelerating through the stop sign, V; the distance between the vehicle centerline and middle of the road, L; and the overall steering smoothness, $\int |\delta| dt$. The speed, V, determines whether the driver has either stopped or decreased speed before the stop sign. The distance, L, indicates the position of the vehicle to make sure the vehicle is not driving off the road or across the double yellow line. A smaller steering angle, δ, while driving through this scenario is required; $\int|\delta|dt$ signifies if the driver controls the vehicle smoothly. Table 2.1 lists the values for α_{1j} and $\overline{K_{1j}}$; each $\overline{K_{1j}}$ relates to a points range (between 0 and 4) depending on different requirements that the driver has attained.

Table 2.1: Scoring Metric for “Traffic Control Device” Module (i=1) with α_{1j} and $\overline{K_{1j}}$

j	α_{1j}	$\overline{K_{1j}}$				
		0	1	2	3	4
1	15	$V>32$km/h	24km/h$<V\leq$32km/h	16km/h$<V\leq$24km/h	8km/h$<V\leq$16km/h	$V\leq$8km/h
2	5	na	na	$L>3$m	$L<0$m	0m$\leq L\leq$3m
3	5	$\int\lvert\delta\rvert dt>360°$	$270°<\int\lvert\delta\rvert dt\leq360°$	$180°<\int\lvert\delta\rvert dt\leq270°$	$90°<\int\lvert\delta\rvert dt\leq180°$	$\int\lvert\delta\rvert dt\leq90°$

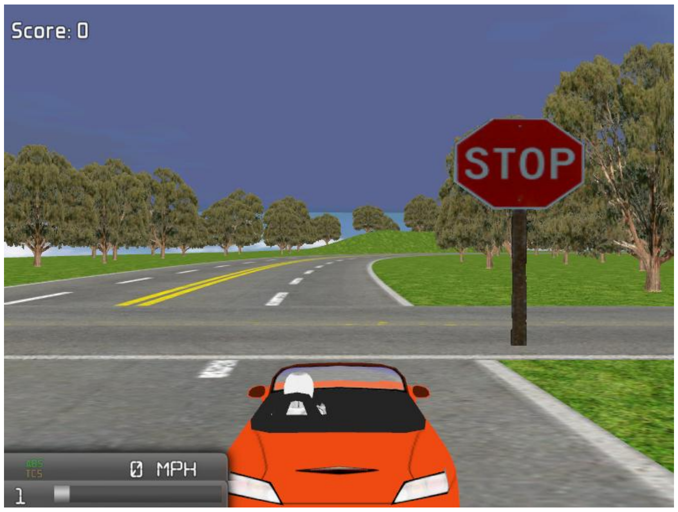


Figure 2.3: Stop sign on the roadway during the “traffic control device" module (i=1)

<u>2.3.2 “Lane Selection" Module</u>

The second module, lane selection, was introduced to analyze how drivers react to roadway obstacles. Obstacles often appear on highways in real life due to road maintenance and/or vehicle crashes which require the driver to quickly change lanes. In this scenario, an arrow sign has been placed in the middle of the road. Behind the arrow sign there are some obstacles that separate the road into two sides - left and right. Since the right lane is “under construction", the arrow points to the left, directing the driver into the left lane. The driver should recognize the road sign and comply. After the driver passes the signs and obstacle, a score will display on the screen. Table 2.2 lists the parameters used to evaluate the driving performance and their contribution to the score. The peak velocity, Vp, reflects whether the driver slowed down sufficiently for the scenario. Driving on the left lane or right lane, F, indicates whether the driver obeyed the sign. The overall smoothness, $\int|\delta|dt$ revealed whether the driver steered the vehicle smoothly.

Table 2.2: Scoring metric for “Lane Selection” module (i=2) with α2j and $\overline{K_{2j}}$

j	α_{2j}	$\overline{K_{2j}}$				
		0	1	2	3	4
1	5	$V_p > 96$km/h	80km/h$< V_p \le 96$km/h	64km/h$< V_p \le 80$km/h	48km/h$< V_p \le 64$km/h	$V_p \le 48$ km/h
2	15	F=0	na	na	na	F=1
3	5	$\int\lvert\delta\rvert dt > 360°$	$270° < \int\lvert\delta\rvert dt \le 360°$	$180° < \int\lvert\delta\rvert dt \le 270°$	$90° < \int\lvert\delta\rvert dt \le 180°$	$\int\lvert\delta\rvert dt \le 90°$

2.3.3 “Panic Braking” Module

The third module is called the “panic braking” module. On the highway, most “rear-end” crashes are caused by drivers who follow too closely behind the lead vehicle and/or neglect to pay proper attention to the roadway. Further, novice drivers have a tendency to over-estimate their vehicles' braking system and their reaction times for engaging the brake pedal in an emergency situation. Thus, a braking module has been designed which requires the driver to complete several steps. The driver will see a posted speed limit sign with higher speed on the screen asking the driver to speed their car up to 96 km/h. Next, when the speed has reached the speed limit, a “panic stop” image will appear on the screen; at this moment, the driver should begin to stop the car immediately. After the car has come to a halt, a score appears on the screen. Table 2.3 lists the detailed items used to evaluate the driver's performance on this scenario. Based on these items, the driver will either pass or fail the task. The first item in Table 3, F, indicates whether the driver stopped completely after seeing the panic stop sign. An F = 1 indicates that they successfully stopped, while an F = 0 denotes that they failed to stop. Reaction time, R, is the amount of time requires for the driver to hit the brake pedal after seeing the stop sign. Braking distance, D, measures the distance from the point at which the driver started pressing the brake pedal to the point at which the vehicle completely stopped. Braking time, T, records the time period between when the driver pressed the brake pedal and the vehicle comes to a complete stop. The deceleration, A, details the amount of G-force the driver would experience when braking the vehicle; usually the deceleration ranges between $0.7 \le A \le 0.9$. The equation $u = |A-0.8|$ represents this behavior. The overall steering smoothness, $\int|\delta|dt$ offers an overview if the driver steered the vehicle smoothly. Finally, the lateral position, L, shows the distance between the vehicle centerline and the double yellow line.

Table 2.3: Scoring metric for "Panic Braking" module (i=3) with α_{3j} and $\overline{K_{3j}}$

j	α_{3j}	$\overline{K_{3j}}$				
		0	1	2	3	4
1	7	F=0	na	na	na	F=1
2	3	R>2.5sec	2.0sec<R≤2.5sec	1.5sec<R≤2.0sec	1.0sec<R≤1.5sec	R≤1.0sec
3	3	D>60m	45m<D≤60m	30m<D≤45m	21m<D≤30m	D≤21m
4	3	T>4.0sec	3.0sec<T≤4.0sec	2.5sec<T≤3.0sec	2.0sec<T≤2.5sec	T≤2.0sec
5	3	$u>1.0m/s^2$	$0.5m/s^2<u\le1.0m/s^2$	$0.3m/s^2<u\le0.5m/s^2$	$0.1m/s^2<u\le0.3m/s^2$	$u\le0.1m/s^2$
6	3	$\int\|\delta\|dt>360°$	$270°<\int\|\delta\|dt\le360°$	$180°<\int\|\delta\|dt\le270°$	$90°<\int\|\delta\|dt\le180°$	$\int\|\delta\|dt\le90°$
7	3	0m	0m	L<0m	L>3m	0m≤L≤3m

2.3.4 "Obstacle Avoidance" Module

The fourth, obstacle module, simulates common roadway obstructions such as an animal crossing the road, a fallen tree, or trash. A crash will occur if the car fails to properly avoid these objects. The module was designed to help train the driver to identify, prepare, and execute correctly in this type of emergency situation. The driver should recognize the obstacle and respond immediately to avoid hitting the object on the road. A drivers' performance will be evaluated based on three factors (refer to Table 2.4). The speed when passing the animal, V, indicates whether the driver has slowed the vehicle to pass the obstacle. The distance between the vehicle centerline and the middle of the road, L, detects if the driver avoided hitting the animal successfully. Overall steering smoothness $\int|\delta|dt$ is another factor used to assess the behavior of the driver. For example, turning the steering wheel sharply is dangerous when the driver approaches the animal in the road.

Table 2.4: Scoring metric for "Obstacle Avoidance" module (i=4) with α_{4j} and $\overline{K_{4j}}$

j	α_{4j}	$\bar{K}_{4j}$				
		0	1	2	3	4
1	5	V>80km/h	64km/h<V≤80km/h	48km/h<V≤64km/h	32km/h<V≤48km/h	V≤32km/h
2	15	na	na	L>3m	L<0m	0m≤V≤3m
3	5	$\int\lvert\delta\rvert dt>360°$	$270°<\int\lvert\delta\rvert dt\le 360°$	$180°<\int\lvert\delta\rvert dt\le 270°$	$90°<\int\lvert\delta\rvert dt\le 180°$	$\int\lvert\delta\rvert dt\le 90°$

2.3.5 Data Analysis

After the driver has finished two laps on the track, data collected such as the vehicle position (x, y) and steering wheel angle, can also be analyzed to visually evaluate the drivers' performances. Figure 2.4(a) shows the track position for an experienced driver, which looks quite smooth without abrupt changes with the motile matching to the actual track. In contrast, Figure 2.4(b) shows a path driven by a novice driver. In this case, there are very few smooth lines and each corner of the track cannot be recognized clearly. As a result, it is obvious that a good driver performs much better on the simulation. A good driver is able to maintain smooth control of his car and steer accurately, while the bad driver struggles to control the car.

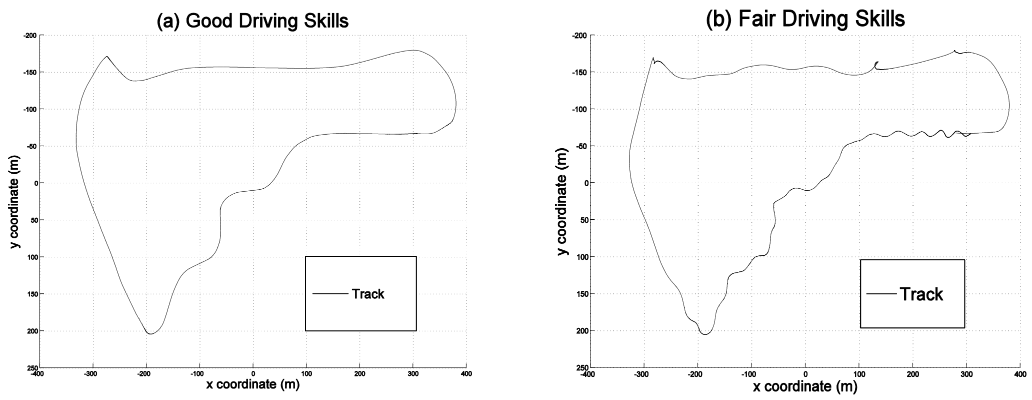


Figure 2.4: Representative car trajectory as measured by (x-y) coordinates with a comparison of two drivers with skills rated as good (a) and fair (b), respectively

Monitoring the steering angle can help evaluate whether the student has driven their car smoothly. If the steering angles are large, then the driver turns the wheel frequently; this typically indicates poor driving performance. If the steering angle is small, then the driver does not rapidly turn the wheel, which often signifies a good driver. Figure 2.5(a) displays the steering angle for an experienced driver which resides between $-60° \leq \delta \leq 60°$ with linear changes, which means the driver does not turn the steering wheel more than 90°. Figure 2.5(b) records the steering angle for a novice driver, in which case the steering angle is between $-450° \leq \delta \leq 450°$. The driver turns his steering wheel excessively which indicates poor vehicle control. In short, a good driver can turn his car smoothly with a smaller steering angle degree while a novice turns the steering wheel sharply, trying to keep his car on the track.

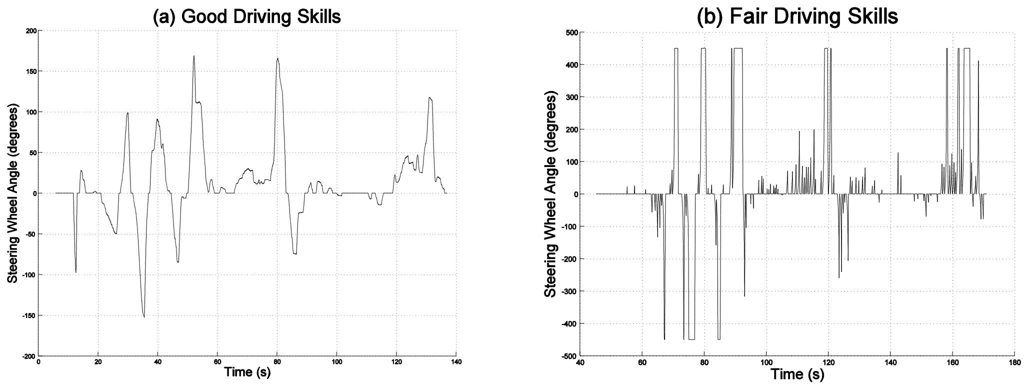


Figure 2.5: Representative steering wheel angles as measured by the data acquisition system for two drivers with skills rated as good (a) and fair (b), respectively

2.4 Pilot Study to Evaluate the Driving Simulator

A pilot study was completed to verify the feasibility of the driving simulator for use in novice driver training. A total of 12 Clemson University students were invited to complete the initial testing of CATS. The students were between 23 and 27 years old which corresponds to an at risk population group (Christian *et al.*, 2012). Nine of the students (all males) had over two years of driving experience. Drivers were first allowed to drive on a simple track to gain the feel of the simulator. When they felt comfortable, the drivers completed the pre-simulation questionnaire. It contains 10 questions; five basic driving knowledge questions worth 20 points and five driving attitude questions not scored. The total score is 100 points. The driver then completes a two-lap drive on the simulator. Before each of the four scenarios, information will be displayed on the screen to inform the driver of the situation ahead. After the driver has passed each scenario, a simple summary was shown on the screen to inform the driver of their performance. After the first lap, driving rate information was displayed on the computer screen. The driver could return to the main menu to find specific scores for each scenario by looking under the score menu. The driver then watched a five-minute instructional video on how to drive correctly in order to pass each scenario and afterwards drove the same track again. This time, the score was used to rate their driving performance.

The complete pilot study data is summarized in Table 2.5. One-third of the drivers had held their driver's license for less than two years and do not have any formal driving education experience. However, two-thirds of all the drivers rated their skills above average, and only one driver rated his driving skill as fair. For the pre- and post-simulation questionnaire, five drivers received a perfect score, four drivers increased their knowledge scores, and two drivers' knowledge scores decreased. Overall, a 5% increase was realized after the drivers completed the driving simulator training. The results of the driving scores for the first run and second run have also been listed in the Table 5. Six of the twelve drivers scored lower on the second run of the simulator than on the first run. For five of these six drivers, the difference between the scores of the first and second run was less than 5%, indicating that the driving simulator did not improve their performance. The remaining drivers scored 20% lower on the second run than on the first run, and performed notably well on the first run. The driver in question had three years of driving experience and had completed a driver's education course. His exceptional performance on the first run testifies to his ability as a competent driver; however, on the simulation he tended to drive aggressively and test the limits of the vehicle dynamics after familiarizing himself with the scenarios presented in the simulation. This behavior can account for his low score on the second run. The other six drivers scored higher on the second run than on the first one; four of them scored 18% higher while two of them scored 30% higher. The overall results are an average increase in driving performance from the first run to the second run.

The drivers' ratings are based on the scores of the second run, and are consistently lower than the driver's self-ratings. This discrepancy suggests that many drivers overestimate their driving skills. The data for two subjects (2, 12) are highlighted in Table 2.5. Although both drivers are 25 years old, they have drastically different levels of driving experience (0-2 years vs. 6-10 years) and both scored higher on the second run than on the first run. Subject 2 rates himself as an “excellent” driver while the simulation rates him as merely a “good” driver. While the score is in fact “good”, subject 2 exhibits over-confidence in his driving ability. Subject 12 is a novice driver with less than 2 years of driving experience and no background in driver's education. His driving scores increased by 13 points, indicating a marked increase in driving performance. In both cases, the driver performance benefitted from the automotive simulator exercises.

Table 2.5: Pilot study results for twelve human test subjects using the CATS system

Subject	Age	Years Driving	Self Rating	Driving Education	Pre Test Score	1st Run Driving Score (DS)	2nd Run Driving Score (DS)	Driver Rating (DR)	Post Test Score	Knowledge Gain (%)
1	27	3-5	Excellent	Yes	100	86.75	66	Fair	100	0
2	25	6-10	Excellent	No	100	61.25	80	Good	100	0
3	21	6-10	Good	Yes	80	84.75	80	Good	100	20
4	24	3-5	Good	No	60	45	47.5	Dangerous	80	20
5	24	6-10	Excellent	Yes	100	54.75	61.25	Fair	100	0
6	23	0-2	Average	Yes	60	59.25	55	Dangerous	100	40
7	26	6-10	Average	Yes	100	79.25	76	Average	100	0
8	28	11-20	Good	Yes	80	74.75	73.75	Average	100	20
9	23	0-2	Average	Yes	100	62.25	58.5	Dangerous	80	-20
10	27	3-5	Good	Yes	100	19	54.25	Dangerous	80	-20
11	23	0-2	Fair	No	80	21.5	65.5	Fair	80	0
12	25	0-2	Good	No	100	71.25	84.25	Good	100	0
Average	24.67	5.25	Good	-	88.33	59.98	66.83	Fair	93.33	5

2.5 SUMMARY

Driving a vehicle is a dangerous undertaking for all drivers; especially for novice drivers which represents the highest source of injury and death for this age group. To address this problem, a driving simulator education system was developed for teaching new drivers how to drive safely. Twelve individuals were invited to participate to a small pilot study. The participants answered ten pre-test questions before driving the simulator, and then ten post-test questions after driving the simulator. In the simulator, they drove the same lap twice. While they were driving, data was collected and used to rate their driving performances. Results show that their driving rate improved on the second lap by 6.8%, which suggests that this driving simulator can help improve novice drivers' performances. After each simulation run, a detailed table appeared on the screen, providing insightful information on which skills the drivers may need to practice. The track and its features have been modified based on suggestions by the participants of the pilot study which will facilitate the comprehensive large scale case study.

CHAPTER THREE

ASSESSMENT OF AN AUTOMOTIVE DRIVING SIMULATOR TO EDUCATE NOVICE DRIVERS

Novice drivers are more likely to be involved in vehicle crashes than more experienced drivers. Transportation system simulators, such as aerospace, automotive, and rail, have been used effectively over several decades in support of operator training and research studies. The immersive high-end automotive simulators tend to be large-scale, difficult to move, and relatively cost ineffective for widespread deployment. In contrast, a mobile vehicle training system often features simplistic environments which are not readily accepted by teenagers who evaluate the driving experience as compared to commercial video games. Consequently, a need exists for a low-cost portable automotive training system that provides a higher level of realism with superior graphics for novice drivers. In this paper, a turn-key computer system is presented to assist novice drivers in the improvement of their driving skills with performance evaluations. The custom simulator was assessed by 50 participants who answered pre-test and post-test questionnaires and drove the simulated vehicle around a preset course. The participants overall driving score improved 28% with gains on both driving knowledge (questionnaire) and proficiency (test track). The simulator has been proven to be efficient at raising both driving knowledge and skills; the tool set represents an important resource for driver training programs. The next step will be its integration into a nationwide safe driving program to complement in-vehicle instruction with virtual instruction.

3.1 Introduction

Young drivers, particularly those between the ages of 16 and 23, have more than double the number of car accidents than older drivers (McCartt *et al.*, 2009). Researchers have identified several factors that may cause novice drivers to have more crashes than experienced drivers. First, young drivers have less experience operating vehicles on the roadways and usually overestimate their driving skills (Craen *et al.*, 2011). Second, most young drivers are not as familiar with traffic laws since their past experiences were largely derived as passengers (Dols *et al.*, 2001). Third, novice drivers may inaccurately gauge the speed of cars around them and/or the relative distance between surrounding vehicles (Chan *et al.*, 2010 and Scialfa *et al.*, 2011). Unfortunately, this inexperience may cause them to suddenly decelerate their vehicles and/or change lanes without noticing approaching cars. Finally, young drivers may be more easily distracted by various factors such as roadside advertisements, electronic devices including MP3 players, cell phones, and portable video games, which draw their attention away from the roadway (Green, 2010). Therefore, driver education programs should help novice drivers understand the danger of inexperience and gain critical knowledge and skills through focused classroom, in-vehicle, and simulated instructions.

Driver training laws and programs have existed for many years around the world. The Graduated Driving License (GDL) Program, first introduced in 1996 in Florida, has been adopted throughout the United States in various forms (Ferguson, 2003). The GDL program attempts to increase young driver safety while decreasing their crash rates through restricted motor vehicle operation. Young drivers need to initially pass the standard written and in-vehicle driving tests. The written test requires drivers to answer basic questions designed to evaluate their knowledge of proper vehicle operation, while the vehicle driving test assesses the driver's mastery of fundamental driving skills. However, a GDL program license imposes age-based restrictions on motor vehicle operation. For example, the driver may only operate their vehicle at certain times and without passengers below a prescribed age. As the young driver matures, these restrictions are decreased until the individual is fully licensed. Although the current GDL program has been successful in educating novice drivers, a number of limitations exist with the overall driver licensing process. First, young drivers typically learn how to operate motor vehicles from their parents; so the teaching effectiveness depends on the parents' skills. Second, the driving proficiency test commonly occurs during daytime hours on local roads with minimal traffic, so the driver's proficiency is not fully evaluated on all roads or during nighttime. Consequently, a need exists to better train novice drivers.

Virtual driving education has the potential to become an important tool for training young drivers. Racing games are one type of a virtual driving simulator which has flourished over the past two decades. Some video games, such as "Grand Turismo 5" and "Need For Speed", are quite adept at mimicking real world conditions and pose the question of whether it would be possible to educate novice drivers using immersive automotive simulators. Young drivers are more likely to spend time practicing a driving skill if it is also delivered in an entertaining manner (Wahlberg, 2010). Furthermore, a greater variety of driving scenarios can be created within a simulation environment much easier than they can be implemented at a test track. The scenarios implemented in a simulator are also inherently safer than their real world equivalents and allow risky scenarios such as two wheels off, excessive speeding, driving too close to a lead vehicle, using a mobile phone while driving (including text messaging), and violating traffic rules. Based on this concept, driving simulators designed specifically for training young drivers have been developed (Kemeny, 2003). Through the efforts of the researchers, virtual driving simulators to train young drivers have become more effective and practical (Park and Lim, 2003). Norfleet *et al.* (2009) applied a large scale virtual driving simulator to improve driving skills. However, since the simulator was fixed in a psychology laboratory, it was not practical for training people off the Clemson University campus. Thus, it became apparent that a more portable version of the driving simulator would be ideal.

The Clemson Automotive Training System (CATS), a portable driving simulator (shown in Figure 3.1), has been developed to help improve driving experiences and reduce potentially dangerous behaviors (Yao *et al.*, 2013). The training system combines three aspects: a pre-test questionnaire, four driver modules, and a post-test questionnaire. First, the drivers answer ten multiple choice questions prior to driving. Second, they drive on a virtual test track, in which four modules evaluate different aspects of the drivers' behaviors and reactions to various scenarios. An assessment system records different types of data in the background to identify and analyze the drivers' performances. Third, the driver completes a post-test questionnaire after completing the modules. Both the pre-test and post-test questionnaires are recorded and compared. The scores of the first and the second driving runs are combined to create a total score. A total of 50 people have participated in this study and the results have shown an overall score improvement of 28% in their driving performance.

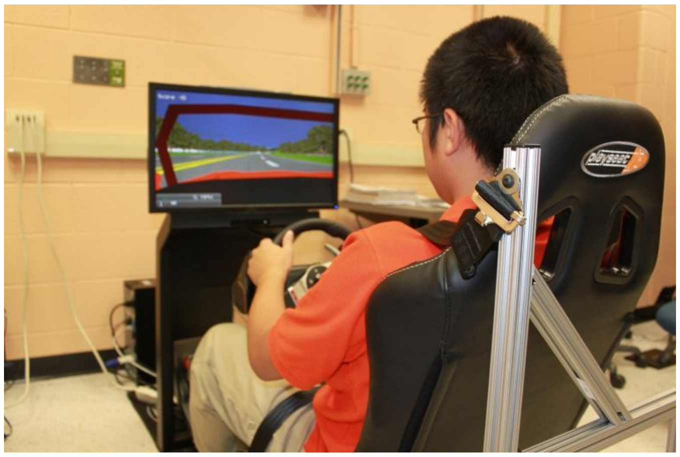


Figure 3.1: Portable Clemson Automotive Training System (CATS) with seat belt, standard human / vehicle interface, screen display, speaker and the host PC workstation

The remainder of this paper is organized as follows: Section 2 introduces the Clemson Automotive Training System, including the hardware, software, features, and track scenarios. Section 3 describes the process of completing the virtual driving test and analyzing the data. Section 4 contains the conclusion. The six Appendices present the human test subjects' results (Appendix A and B), demographic questionnaire (Appendix C), the scoring metrics (Appendix D), and Nomenclature list (Appendix E).

3.2 Clemson Automotive Training System

The Clemson Automotive Training System was assembled using a desktop computer with a 1GB graphic card, a 19 inch monitor with a sound bar, a racing chair that can be adjusted vertically and horizontally, and a Logitech G27 steering wheel with 900 degrees of rotation (refer to Figure 3.2). The program was coded using C++ and Python, and it is executed in Ubuntu 10.04 (a Linux operating system). The driving track in this simulator is highly customized; the track was designed and created using Bob's Track Builder (BTB, 2012) and Blender (Roosendaal, 2012). The simulator was developed based on VDrift (Venzon, 2012). The created user interface allows the driver to read information and view images on the screen during their driving session. One item always displayed on the top left of the screen is the participant's driving score which allows them to receive immediate feedback about their driving performance. In addition, a detailed scoring system rates the user's driving skills and can be viewed during and/or after a run to offer in depth feedback. For example, the average velocity, maximum steering angle, and other information are collected and used to analyze the drivers' performances. Finally, several scenarios to evaluate the driver's proficiency were created.

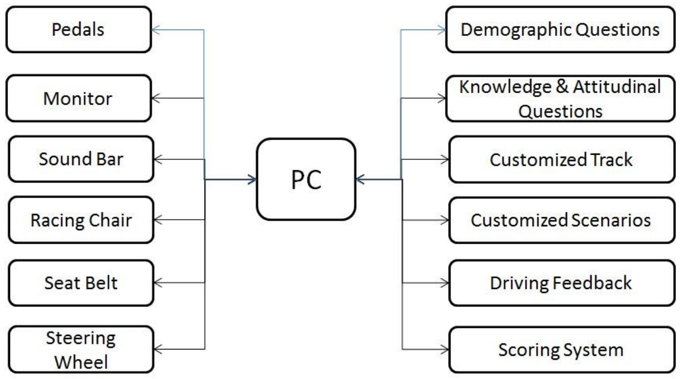


Figure 3.2: Hardware and software configuration in CATS

The driving simulator trains young drivers and simultaneously evaluates their driving skills based on four modules. In the first scenario, a stop sign located at an intersection assesses whether the driver will stop their vehicle as directed. The second scenario simulates a blocked lane that tests whether the driver will react appropriately by slowing down and using another lane to avoid the blockage. In the third scenario, the driver must accelerate their vehicle to a high speed and then stop quickly. This scenario is intended to give the driver a sense of their vehicle's limitations, including the vehicle's stopping time and distance, as well as the driver's reaction time when they meet an emergency situation. The fourth scenario simulates an animal jumping onto the road and tests how the drivers will react to the event. After the simulation, the drivers can view their score for each scenario, allowing them to recognize the areas in which they can improve.

3.3 Virtual Driving and Data Analysis – Case Study

A human-subjects experiment was designed to test the effectiveness of the virtual driving education simulator to train and educate novice drivers. A total of 50 subjects participated in this study. Among these subjects, 35 participants were male and 15 participants were female. These participants' ages ranged between 23 years old and 31 years old. They were classified as novice (0-2 years), beginner (3-5 years), and experienced (5+ years) drivers based on their years of driving. These subjects were invited to participate and complete the seven steps driver training process. These seven steps included: (a) read the policy, sign a consent form, and complete a written questionnaire asking basic information (e.g. age, gender, driving habits, etc. as listed in Appendix C); (b) sit in the racing chair, buckle the seat belt, and read the instructions to familiarize themselves with the equipment; (c) drive for five minutes on a practice track to learn the throttle, brake, and steering wheel settings; (d) drive on the track for one lap without directions (although, they were able to see the final score after completing this lap); (e) watch a short video indicating the correct method for driving and passing each scenario correctly; (f) drive on the same track for a second time; and (g) review the Excel file showing their detailed driving information and their performances.

3.3.1 Pre-Test and Post-Test Data Analysis

A ten question questionnaire, divided into two parts – attitudinal and knowledge, was created for the participants to answer before and after their virtual driving test. The attitudinal questions evaluate the subjects' driving attitude to detect potentially risky driving habits. Table 3.1 exhibits the responses to the five questions where SD, D, A, and SA correspond to strongly disagree, disagree, agree, and strongly agree, respectively. The results of both the pre-test and post-test questionnaire are displayed with the overall percentage improvement of the driving education program.

The first set contains five attitudinal questions. Each question has a preferred answer which has been marked with a star. Question Q1 determines whether the subjects are likely to show off their driving skills. Over 80% of test subjects selected disagree or strongly disagree, indicating that they are unlikely to show off while driving. Question Q2 determines the percentage of participants that tend to drive over the speed limit. The results show that over 40% subjects think that speeding is ok and, even after the driving test, 36% of the subjects maintained their original attitude. This information indicates that a large proportion of drivers do not realize the potential risks created by exceeding the speed limit. Question Q6 asks the participant whether wearing a seatbelt makes them feel safe; more than 90% subjects felt safer wearing a seat belt which indicates that the subjects realize the safety benefit of wearing a seat belt. Question Q10 is designed to find out whether the participants would be willing to ride in a car with a potentially unsafe driver if they had no other way home. Unfortunately, 30% of subjects indicated that they would in fact ride with an unsafe or reckless driver.

Table 3.1: Result of pre-test and post-test attitudinal questions on questionnaire (SA: Strongly agree, A: Agree, D: Disagree, SD: Strongly disagree)

Attitude Question	PreTest (%)				PostTest (%)				Improvement (%)
	SA	A	D	SD	SA	A	D	SD	
Q1: I love to show offwhen I'm driving	4	16	50	30*	6	10	38	46*	16
Q2: If you have good skills, speeding is OK	8	36	34	22*	12	24	34	30*	8
Q6: Wearing a seatbelt makes me feel safe	68*	28	4	0	62*	30	2	6	-6
Q8: I'm still learning to be a good driver	34*	44	18	4	40*	44	16	0	6
Q10: I would get into the car with a reckless driver if I had no other way to get home	6	28	36	30*	6	24	30	40*	10

The second set of five multiple choice questions were designed to evaluate the subjects' driving knowledge in terms of proper driving techniques, traffic signs, and safe driving practices. The questions and both pre-test and post-test results are listed in Table 3.2. The columns labeled "correct" and "incorrect" show the percentage of subjects that answered the question correctly and incorrectly. The column labeled as "Improvement" shows the percentage of subjects that improved their answer after taking the driving test. Question Q3 evaluates whether each subject knows the right way to wear a seat belt. The results show that 72% answered correctly on a pre-test and 92% on a post-test. A 20% improvement is significant. Question Q4 tests whether the subjects understand how to properly check their mirrors while driving and 95% answered correctly, showing that nearly all of them are aware of this safe driving practice. Question Q5 tests the drivers' reactions upon encountering a red light on the road; 74% got the correct answer on the pre-test and 90% of the subjects answered correctly on the post-test. Question Q7 asks when a driver should use a turn signal on the express way. Over 90% of the subjects answered correctly on both the pre- and post-tests. Question Q9 evaluates if the subjects know the correct response to a flashing yellow light at an intersection, specifically when

Table 3.2: Result of pre-test and post-test knowledge questions on questionnaire

Question	Pre-Test (%)		Post-Test (%)		Improvement (%)
	Correct	Incorrect	Correct	Incorrect	
Q3	72	28	92	8	20
Q4	94	6	96	4	2
Q5	74	26	90	10	16
Q7	90	10	96	4	6
Q9	66	34	82	18	16
Q3:	What is the proper way a seat belt should be worn? *a) Cross the shoulder and chest, across the hips and upper thigh b) Across the chest and waist c) Not at all d) Below rear and across waist				
Q4:	When driving, you should consistently check the following? *a) Road, mirrors and traffic b) Side mirrors c) Rear view mirror d) The road				
Q5:	Coming to a flashing red light, what should you do? a) No action required, maintain speed b) Continue through without stopping if no vehicles are around c) Slow down, checking for traffic before proceeding *d) Come to a complete stop and look for traffic				
Q7:	While driving on a highway, when do you use your turn signal? a) It is not necessary to use a turn signal to change lanes b) Only in heavy traffic and at night time, use your turn signal while changing lanes *c) Use your turn signal 50 yards before changing lanes d) Using a turn signal only in reduced visibility conditions				
Q9:	When approaching an intersection with a yellow signal light, it is best to… a) Coast until the light changes back to green *b) Come to a complete stop when red, turn right on red when safe to proceed c) Slow down to a complete stop and then turn right d) Speed up and travel through intersection				

turning right; 66% of the subjects answered correctly on the pre-test and 82% on the post-test. The overall improvement indicates that after the subjects finished the driving test, they demonstrated a significant improvement in their understanding of driving knowledge and techniques.

<u>3.3.2 Methodology for the Driving Score and Driving Rating</u>

An algorithm has been created to access the driver's performance by calculating the driving score (DS) and the driving rating (DR), after they completed the program. The score of each module, S_i for (i = 1, 2, 3, 4) corresponds to each driving feature and is expanded as

$$\mathbf{S}_i = \left\{ \begin{array}{ll} \sum_{j=7}^{3} \alpha_{ij} \overline{\mathbf{K}_{ij}}; & \text{for}(i=1,2,4) \\ \sum_{j=1}^{7} \alpha_{ij} \overline{\mathbf{K}_{ij}}; & \text{for}(i=3) \end{array} \right\} \tag{3.1}$$

where i represents the scenario, j represents the items that calculate scores in each scenario, and $\overline{K_{ij}}$ is the metric score, α_{ij} is a weight indicating the importance of items in each scenario. The values for $\alpha_{ij}\overline{K_{ij}}$ have been defined in Appendix D and the reader is referenced to [12] for more information.

The overall driving score, which reflects the individual driving score on each module as well as difficulties with speeding, driving off the road, and driving across the double yellow line, maybe calculated as

$$\mathbf{DS} = \frac{1}{n}\left(\sum_{i=1}^{4} S_i - \sum_{\kappa=1}^{3} \beta_\kappa N_\kappa \right) \tag{3.2}$$

where n=4 is used to normalize the final score between 0 and 100, *k* represents the number of N factors, and $\beta=\{20, 20, 20\}^T$ is a weight indicating the importance of each N_k. The difference value comes as a result of a comparison of the first lap and second lap. If the difference value is positive, then the subject improved on their driving skills with the driving simulation.

The driving rating (DR), is based on the driving score (DS), to show a subject's driving performance using the following mapping

$$\mathrm{DR} = \begin{cases} \text{Excellent}; & \text{if } 90 \le \mathrm{DS} \le 100 \\ \text{Good}; & \text{if } 80 \le \mathrm{DS} < 90 \\ \text{Average}; & \text{if } 70 \le \mathrm{DS} < 80 \\ \text{Fair}; & \text{if } 60 \le \mathrm{DS} < 70 \\ \text{Dangerous}; & \text{if } \mathrm{DS} < 60 \end{cases} \tag{3.3}$$

3.3.3 Data Analyzing for Test Subjects Using the CATS System

The drivers' performances were evaluated after they finished driving the simulator tracks. In Appendix B, the average driving score on the first lap was DS = 57 which reflects a dangerous driver. After subjects became familiar with the road and driving scenarios, the average driving score increased on the second lap to DS = 75. This corresponds to an average driver rating as 28% improvement.

The driving scores of 11 subjects reflected a lower second run driving scores than their first run. Subjects 4, 14, and 34 achieved the same scores between the first run and the second run; all achieved good driver's ratings which shows that they had limited need to be trained and educated by CATS to improve their driving skills. Lastly, 36 subjects improved their driving scores ranging from 3% to 84.5%. The average improvement of the 50 subjects was 28%. Subject 2 is highlighted because he was a novice driver whose driving experience was less than two years. His first run score was 66.8; this represents his fair driving skills. His second run score was 80.8, which represents his good driving skills. His overall improvement was 21%; this demonstrates that CATS is useful for helping novice drivers become familiar with real road conditions and improving their driving ability.

One aspect of the study evaluates the testing improvements in novice versus experienced due to the driver training system. Between drivers of different skill levels, novice drivers improved more than experienced drivers. This seems to be explained by the fact that the majority of experienced drivers already know most traffic rules; while novice drivers are not as familiar with traffic rules and therefore, may not have the proper reactions to pass the scenarios in the CATS system. Novice drivers need more practice in this type of simulation which can improve their driving skills through the program such as CATS. Novice drivers are a major group whose driving performances increased sharply. Both novice and beginner drivers improved their driving scores over 25%, demonstrating that CATS is effective in improving student drivers' performances.

3.3.4 Data Analysis for Testing Scors and Other Factors by Classification

Five factors, including peak velocity, V_{peak}, average velocity, V_{ave}, peak steering angle, δ_{peak}, and average steering angle, δ_{ave}, were found to influence the students' driving performance (refer to Appendix C). Higher testing scores correlated with lower maximum velocities. It is harder for drivers to react properly to some situations when the vehicle is moving at a faster speed. A faster speed also corresponds to a higher frequency of driving off roads or crossing double yellow lines when driving on a curvy road. Compared to the first run, the second run's V_{peak} decreased by 3% while V_{ave} increased by 6%, and the DS increased by 28%. V_{peak} decrease suggests that increased driving safety occurred when decreasing the speed. V_{ave} increase shows that after the driver becomes familiar with the track, they can drive more safely. Moreover, two factors: the maximum steering angle, δ_{peak} and the δ_{ave} share the similar trend with the velocity. The lower these three factors, the higher the driving score. In addition, between the first and second run, both factors decreased, with δ_{peak} decreasing the most with a drop of 40%. Steering angle represents the driver's ability to drive the vehicle smoothly; the smaller the steering angle, the better one can control the vehicle. The decreased number of crossing double yellow lines, N_1, driving off roads, N_2 and driving over the speed limit, N_3 indicates that the drivers became more cautious of driving on the curvy road and they demonstrated a better awareness of controlling vehicle speed under the limit compared to their first run. Overall, this illustrates that the drivers improved their driving performance after achieving familiarity with the track and scenarios on the road thanks to the help of the driving simulator.

3.3.5 Data Analysis for Self-Rating and Test-Rating

The results of different groups of people who rated themselves and the driving score they achieved after taking the driving test were analyzed (refer to Table 3.4). The subjects were classified by the number of driver years (DY). The table shows both the self-rating questionnaire and the driver test-rating classified into five parts: excellent, good, average, fair, and dangerous. The result of the score in the self-rating and the driver test-rating shows most novice drivers rated themselves between good and average, while their test ratings are either average or dangerous. Experienced drivers are more confident of their driving skills. Of the nine drivers which were experienced. Four of them achieved a score of good, while the rest of them tended to drive fast in CATS and thus, achieved low driving scores.

Table 3.3: Results of driver self-rating and CATS test-rating among different subject grouping

	Novice (0~2 years)	Beginner (3~5 years)	Experienced (6+ years)
Number of Subjects	20	21	9
Driver's Self Rating on Pre-test Questionnaire (Question 4)			
Excellent	1	4	3
Good	7	9	3
Average	8	5	2
Fair	0	3	1
Dangerous	4	0	0
CATS Driver Rating, DR			
Excellent	0	0	0
Good	9	11	4
Average	2	7	3
Fair	4	3	1
Dangerous	5	0	1

3.4 Summary

CATS offers novice drivers a platform to safely increase their driving skills through continuous virtual monitoring and testing. Several results have been observed. First, novice drivers demonstrate significantly greater potential in enhancing their driving skills and absorbing traffic rules than experienced drivers. Second, a relationship between the drivers' scores and their maximum/average velocity was observed: an increase in driving score correlates with a decrease in peak velocity, but an increase in average velocity. Third, a large proportion of drivers do not realize the potential risks created by driving over the speed limit. Finally, an average improvement of 28% in the driving score indicates that CATS succeeded in improving driver’s simulated performance. The simulator has proven efficient at raising both the driving knowledge and skills; this tool represents an important resource for driver training programs.

CHAPTER FOUR

CONCLUSIONS AND RECOMMENDATIONS

Every year there are a large number of road crashes in the United States. Young drivers have a higher percentage rate of being injured or killed compared to experienced drivers. Many researchers are working on driver training and have found various methods to improve young drivers' driving skills and to improve their ability to react to hazardous situations that occur when they are driving. A virtual driving simulator offers one solution to train young drivers. The Clemson Automotive Training System (CATS) is a tool that can be used to educate and train young drivers, resulting in a significant increase in their ability to safely operate motor vehicles.

4.1 Clemson Automotive Training System and Pilot Study

A portable driving simulator has been developed by Clemson Univeristy Automotive Safety Sesearch Institute (CU ASRI). In this driving simulator, the driving station has been assembled with various components including desktop computer, a 17inch monitor, a racing chair, and a steering wheel with foot pedal controls. A seat belt has been designed and attached to the racing chair. A track has been designed and created using commercial software to simulate a real test track and roadway traffic conditions. Four modules were then implemented in the simulator to test different aspects of the students' driving skills. Also, a driver's feedback system was implemented to offer participants information indicating the right way to pass each module. Additionally, a driving score has been calculated by adding the points the drivers received from each module and displayed both in the vehicle and memory. A database was created to collect a variety of driving parameters to further research these students' driving behaviors and their potential risks on the road.

Initially, twelve students were invited to participate in the virtual driver's test as pilot study to evaluate the practicality and effectiveness of this simulator. The on track driving performance of these students improved an average of 12.75% (novice), 5.67% (young), and 4.31% (seasoned). Similarly, the students displayed an increase between their pre-test and post-test scores which indicated a positive trend for gained knowledge on driving rules. A few areas were pointed out by participants for improvement as following: the unnecessarily complex shape of the track, the location of the feedback information, and the scoring criterion. Based on their feedback, we have made these improvements.

4.2 Assessment of a Driving Simulator for Training Novice Drivers

An assessment of the driving simulator for novice drivers was undertaken to evaluate the effectiveness of this virtual tool set. The assessment consists of two parts: a knowledge questionnaire and driving modules. The questionnaire contains attitudinal and general driving knowledge questions. The attitudinal questions evaluated the driver's attitudes toward proper habits when operating motor vehicles while the general driving knowledge questions tested the drivers on some basic road rules. The second part was the driving simulator, which gives these students an opportunity to drive on a track where the four modules are implemented; each module evaluated certain aspect of their basic driving skills. At the end of this program, a driving rate that correlates to a driving score was recorded to be used to evaluate the driver's performance.

The final assessment evaluated 50 students who were invited to participate in the CATS program. These participants had an overall 28% improvement in driving scores. In addition, these drivers had a noticeable increased between the pre-test and post-test questionnaire. This indicates that these participants have learned more driving knowledge and the right attitude in certain driving situations. The portable CATS has proved to be efficient at improving driving skills and knowledge, and represents an important tool for a drivers training program.

4.3 Recommendations for Future Research

In this project, many different components have been developed. These include: the program to build the track in CATS, designing and implementing modules to the simulator, creating a scoring and a feedback system, building a database for analyzing parameters in the simulator. In addition, CATS has proved to be effective in improving driver's performance. This is a perfect time to consider moving forward to the next development stage. Here are some suggestions for future work.

4.3.1 In-Vehicle Force Feedback

Since Ubuntu, the Linux operating system does not fully support the software code, refer to a software driver for the G27 Logitech steering wheel. The force feedback function cannot be used in the simulator. There are two ways to solve this problem. First, since another racing simulator has fixed this problem by writing a driver which supports force feedback in Linux, it is possible to get open source code, and implement it into

CATS. The second method that could be used is to transplant the whole CATS from the Linux platform to the Windows platform because the driver of G27 has supports for Windows, thus solving the problem.

4.3.2 Database Optimization

The database currently used in this simulator is Excel, which is good for analyzing data. However, it is more functionality to build a database based on MySQL. MySQL can handle the data by sorting and filtering. Furthermore, it can be compatible with C++ as well, giving the developer huge benefits. Thus, CATS could be programed to analyze data directly in the simulator instead of dealing with the data afterwards; this will allow the student to see their detailed data immediately after they have finished driving.

4.3.3 New Track Design and Build

This work has focused on the methodology regarding how to create and transport the track from Bob’s Track Builder, to Blender, to CATS. The next step is to create a more realistic track, which can simulate a real environment such as Clemson’s campus, downtown, or a real track in the state. In this manner, students can practice driving in the simulator world while also becoming familiar with real road situations.

4.3.4 Interaction with Artifitial Intelligence (AI) Vehicles

To simulate the real road, it is crucial to support traffic road conditions for drivers. An AI vehicle is a good option. It can support up to four vehicles concurrently driving on the road; moreover, a vehicle driving in the opposite direction could also be programed into the course by changing codes in the track building stage.

4.3.5 User Interface Development

The current project only supports answering questions, running CATS, and watching instructional video in a terminal environment. Currently in the CATS system, a person should stay with the student at all times to input commands and give detailed instructions on how to operate the motor vehicle. A graphic user interface could be useful to solve this problem. A main menu could offer students several options, such as: answering questions, reading instructions on how to drive, watching instructional video, and running CATS. Students then could operate a simulator by themselves with no tutor. A “Tkinter” package, written in Python, is strongly recommended. Tkinter is easy to use and compatible with the Linux operating system. Python is a script language that can handle multiple programs at the same time.

4.3.6 Assessment Strategies

A better assessment strategy is needed. The current project focused on the realization of the CATS and tested its efficiency at educating novice drivers with no specific designed assessment strategy. Since many students get better scores the second time due to their remembering the moment when the test module appears, it is not possible to test if the students really understand the driving rules or just remember the module. Randomized test modules should be implemented into the CATS, so that each lap the student driver will meet unexpected test modules to evaluate their driving performance. This would in a more accurate result compared to the current project since there are only four fixed test modules.

APPENDICES

Appendix A

Pre and Post Test Questions in Safe Driving Program

Question 1: I love to show off when I'm driving.
a) Strongly Agree
b) Agree
c) Disagree
d) Strongly disagree

Question 2: If you have good skills, speeding is O.K.
a) Strongly Agree
b) Agree
c) Disagree
d) Strongly disagree

Question 3: What is the proper way a seat belt should be worn ?
a) Across the shoulder and chest, across the hips and upper thigh
b) Across the chest and waist
c) Not at all
d) Below rear and across waist

Question 4: When driving, you should consistently check what?
a) Road, mirrors and traffic
b) Side mirrors
c) Rear view mirror
d) The road

Question 5: Coming to a flashing red light, you should?
a) No action required, maintain speed
b) Continue through without stopping if no vehicles are around
c) Slow down, checking for traffic before proceeding
d) Come to a complete stop and look for traffic

Question 6: Wearing a seatbelt makes me feel safe.
a) Strongly Agree
b) Agree
c) Disagree
d) Strongly disagree

Question 7: While driving on a highway, when do you use your turn signal?
a) It is not necessary to use a turn signal to change lanes
b) Only in heavy traffic and at night time, use your turn signal while changing lanes
c) Use your turn signal 50 yards before changing lanes

d) Using a turn signal only in reduced visibility conditions

Question 8: I'm still learning to be a good driver.
a) Strongly Agree
b) Agree
c) Disagree
d) Strongly disagree

Question 9: When approaching an intersection with a yellow signal light, it is best to...
a) Coast until the light changes back to green
b) Come to a complete stop when red, turn right on red when safe to proceed
c) Slow down to a complete stop and then turn right
d) Speed up and travel through intersection

Question 10: I would get into the car with a reckless driver if I had no other way to get home.
a) Strongly Agree
b) Agree
c) Disagree
d) Strongly disagree

Appendix B

CATS Results for 50 Human Test Subjects

Subject	Gender	Age	Driving Experience	Driving Score (DS)		Improvement (%)	Driver Rating (DR)
				1st Run	2nd Run		
1	M	25	3~5	46.5	75	28.5	Average
2	M	23	0~2	66.8	80.8	14	Good
3	M	25	0~2	35.2	81.5	46.3	Good
4	M	28	3~5	82	82	0	Good
5	F	23	0~2	73.8	75.5	1.8	Average
6	F	24	0~2	71.3	83.75	12.5	Good
7	F	24	0~2	45	44.5	-0.5	Dangerous
8	M	23	0~2	54.5	73.8	19.3	Average
9	F	26	5~10	74.3	67	-7.3	Fair
10	F	24	5~10	64.5	59	-5.5	Dangerous
11	M	22	5~10	36.3	85	48.8	Good
12	M	20	3~5	39.5	86	46.5	Good
13	M	23	5~10	1.8	77.5	75.8	Average
14	M	20	5~10	86.5	86.5	0	Good
15	M	24	3~5	63.8	77.5	13.8	Average
16	M	27	3~5	91.3	85	-6.3	Good
17	M	25	0~2	5	66.3	61.3	Fair
18	M	27	0~2	26.8	64	37.3	Fair
19	M	23	0~2	30	46.8	16.8	Dangerous
20	M	27	0~2	75.3	69	-6.3	Fair
21	M	26	0~2	47.8	85	37.3	Good
22	M	26	3~5	82.3	84	1.8	Good
23	M	25	3~5	64.3	60	-4.3	Fair
24	M	24	3~5	51.3	88	36.8	Good
25	F	25	3~5	83.8	87	3.3	Good
26	F	24	3~5	66	80.8	14.8	Good
27	F	27	3~5	78.3	66.3	-12	Fair
28	F	26	3~5	68	79.3	11.3	Average
29	M	23	0~2	42.5	54.5	12	Dangerous
30	F	26	0~2	50.5	48.8	-1.8	Dangerous
31	M	25	3~5	-1.3	83.3	84.5	Good
32	M	25	0~2	81.8	81.5	-0.3	Good
33	M	24	0~2	-26.3	50.8	77	Dangerous
34	F	24	3~5	85	88	3	Good

35	M	28	0~2	72.8	84.5	11.8	Good
36	M	24	0~2	65	87	22	Good
37	M	25	3~5	88.3	74	-14.3	Average
38	M	29	3~5	57.5	65	7.5	Fair
39	M	31	3~5	51.3	75	23.8	Average
40	F	27	0~2	74.3	83	8.8	Good
41	F	27	0~2	62	85.8	23.8	Good
42	M	30	5~10	46.3	70	23.8	Average
43	F	25	3~5	72.8	75.3	2.5	Average
44	M	18	3~5	74.3	85	10.8	Good
45	M	32	5~10	50.8	78.8	28	Average
46	M	26	3~5	26.3	76.8	50.5	Average
47	M	26	3~5	72.5	83.3	10.8	Good
48	F	21	5~10	83.8	84.5	0.8	Good
49	M	26	5~10	82.5	80.8	-1.8	Good
50	M	22	0~2	63.8	60	-3.8	Fair
Average				57	75	28	Average

Appendix C

Driving Simulator Study Detailed Results

Subject	Improvement (%)					Improvement (number)		
	Driving Score	**V_{peak}**	**V_{ave}**	**δ_{peak}**	**δ_{ave}**	N_1	N_2	N_3
1	28.5	-5.2	3	-5.2	10.9	0	-1	-3
2	14	-0.6	-0.4	-334.6	-3.3	0	-1	-3
3	46.3	-2.1	0.2	-77.1	34.3	-3	-1	-5
4	0	-0.6	3.3	228.5	8.2	0	0	1
5	1.8	-0.3	1	-268.8	-2.9	-1	0	1
6	12.5	0.2	-2.1	-346.2	-0.4	0	-2	-1
7	-0.5	-0.9	5.2	-279.7	-4.8	-2	-2	3
8	19.3	1	-4.4	-51.4	-8.9	1	-1	-3
9	-7.3	-2.2	2.7	-124.1	2.6	-1	0	2
10	-5.5	-2.8	9.2	-73	11	-1	0	0
11	48.8	-3.8	2.5	-334.6	-18.9	-1	-2	-5
12	46.5	-21.1	1.1	-387	1.3	-1	-3	-6
13	75.8	1.3	-1.7	-247.8	12.6	-1	-4	-6
14	0	-0.2	2.5	65.8	-1	0	0	0
15	13.8	-5.5	-3.2	-296	-7.6	1	-2	-2
16	-6.3	0.1	2	-87.2	-1.6	0	0	0
17	61.3	-10.9	-4.6	0	-14.1	-4	-3	-4
18	37.3	-0.5	0.5	-291.6	-23.1	1	-7	-2
19	16.8	0.5	-4.6	286	0.3	2	-1	-4
20	-6.3	-2.9	6.5	-108.6	7.3	0	-1	2
21	37.3	0.1	6.2	-353.4	-15.9	-2	-2	-2
22	1.8	-1.6	3.1	-71.9	10.1	1	0	0
23	-4.3	-1	1.3	263.4	-2.7	1	0	-1
24	36.8	-3.3	3.7	-90.7	4.7	-2	0	0
25	3.3	-0.9	7.8	5.6	-0.5	0	0	0
26	14.8	-2	0.1	-9.4	-2.6	-1	-2	0
27	-12	1	1.8	7.3	2	-1	0	-1
28	11.3	0.2	3.8	-96.2	0.5	0	-3	0
29	12	-1.5	2.7	-104.5	-0.8	-1	1	-3
30	-1.8	0.9	2.5	109.9	-3.9	2	-3	0
31	84.5	-10.5	-7	-378	-1.3	-3	-4	-9
32	-0.3	-0.7	1.1	-225	-18.9	0	0	0
33	77	-5.4	-0.7	-359.4	-14.5	-2	-2	-5
34	3	1.8	3.2	31.5	1.4	0	0	0

35	11.8	-0.9	1.2	3.1	-1.3	0	0	-2
36	22	-1.5	0.6	31.1	2.9	0	0	-1
37	-14.3	-0.7	1.8	-134.2	-0.7	2	0	1
38	7.5	4.5	2	-18.1	-0.3	0	0	-1
39	20.8	-0.4	-1.9	-322.4	-0.4	-1	-1	-2
40	8.8	-0.6	-1.3	10.2	-0.1	-1	-1	0
41	23.8	0.3	-3.6	-177.4	-1.5	-2	0	-3
42	23.8	-0.9	-1.3	-105.5	0.4	-1	0	-4
43	2.5	-1.9	2.6	185.9	1.5	1	0	-1
44	10.8	-0.1	2.4	15	1.6	-2	0	0
45	28	0.7	1.1	-334.5	18.2	-2	-2	-1
46	50.5	-4.7	-2.1	-213	-1.8	-4	0	-2
47	10.8	-1.7	2.2	-12.8	1.5	-1	0	-1
48	0.8	-1.6	9.4	6.6	4.8	0	0	0
49	-1.8	-1.7	4.8	65.5	12.9	1	0	0
50	-3.8	7.6	-1.5	-146.9	-1.9	1	0	-1
Average	17.2	-1.7	1.3	-103	-0.1	-0.5	-1	-1.5

Appendix D

CATS Demorgraphics Pre-Test Questionnaire

1) Please indicate your gender:

a) Male ☐ b) Female ☐

2) Please indicate your age group:

a) under 18 years old ☐ b) 18-23 years old ☐ c) 24-29 years old ☐

d) 30-49 years old ☐ e) 50-69 years old ☐ f) over 70 years old ☐

3) Please indicate the number of years of your driving experience:

a) 0-2 years ☐ b) 3-5 years ☐ c) 6-10 years ☐ d) 11-20 years ☐
e) 20+ years ☐

4) Please indicate how you would rate yourself as a driver:

a) Excellent ☐ b) Good ☐ c) Average ☐ d) Fair ☐ e) Poor ☐

5) Have you previously received driver's education?

a) Yes ☐ b) No ☐

6) If you answered "Yes" in Question 5), can you please offer some details?

____

7) Do you regularly play any type of motorsports / car racing video games?

a) Yes ☐ b) No ☐

8) If you answered "Yes" in Question 7), can you please offer some details?

____

Appendix E

Scoring Metrics for the Four Modules in the CATS

Module 1: “Traffic Control Device” (i=1)						
j	α_{1j}	$\overline{k_{1j}}$				
		0	1	2	3	4
1	15	V>32km/h	24km/h<V≤32km/h	16km/h<V≤24km/h	8km/h<V≤16km/h	V≤8km/h
2	5	na	na	L>3m	L<0m	0m≤L≤3m
3	5	$\int\|\delta\|dt > 360^\circ$	$270^\circ < \int\|\delta\|dt \le 360^\circ$	$180^\circ < \int\|\delta\|dt \le 270^\circ$	$90^\circ < \int\|\delta\|dt \le 180^\circ$	$\int\|\delta\|dt \le 90^\circ$

Module 2: “Lane Selection” (i=2)						
j	α_{2j}	$\overline{k_{2j}}$				
		0	1	2	3	4
1	5	Vp >96km/h	80km/h< Vp ≤96km/h	64km/h< Vp ≤80km/h	48km/h< Vp ≤64km/h	Vp ≤48 km/h
2	15	F=0	Na	na	na	F=1
3	5	$\int\|\delta\|dt > 360^\circ$	$270^\circ < \int\|\delta\|dt \le 360^\circ$	$180^\circ < \int\|\delta\|dt \le 270^\circ$	$90^\circ < \int\|\delta\|dt \le 180^\circ$	$\int\|\delta\|dt \le 90^\circ$

Module 3: "Panic Braking" (i=3)						
j	α_{3j}	$\overline{k_{3j}}$				
		0	1	2	3	4
1	7	F=0	na	na	na	F=1
2	3	R>2.5sec	2.0sec<R≤2.5sec	1.5sec<R≤2.0sec	1.0sec<R≤1.5sec	R≤1.0sec
3	3	D>60m	45m<D≤60m	30m<D≤45m	21m<D≤30m	D≤21m
4	3	T>4.0sec	3.0sec<T≤4.0sec	2.5sec<T≤3.0sec	2.0sec<T≤2.5sec	T≤2.0sec
5	3	$u>1.0m/s^2$	$0.5m/s^2<u\le1.0m/s^2$	$0.3m/s^2<u\le0.5m/s^2$	$0.1m/s^2<u\le0.3m/s^2$	$u\le0.1m/s^2$
6	3	$\int\|\delta\|dt > 360^\circ$	$270^\circ < \int\|\delta\|dt \le 360^\circ$	$180^\circ < \int\|\delta\|dt \le 270^\circ$	$90^\circ < \int\|\delta\|dt \le 180^\circ$	$\int\|\delta\|dt \le 90^\circ$
7	3	0m	0m	L<0m	L>3m	0m≤L≤3m

Module 4: “Obstacle Avoidance” (i=4)						
j	α_{4j}	$\overline{k_{4j}}$				
		0	1	2	3	4
1	5	V>80km/h	64km/h<V≤80km/h	48km/h<V≤64km/h	32km/h<V≤48km/h	V≤32km/h
2	15	na	na	L>3m	L<0m	0m≤V≤3m
3	5	$\int\|\delta\|dt > 360^\circ$	$270^\circ < \int\|\delta\|dt \le 360^\circ$	$180^\circ < \int\|\delta\|dt \le 270^\circ$	$90^\circ < \int\|\delta\|dt \le 180^\circ$	$\int\|\delta\|dt \le 90^\circ$

Appendix F

Participant User Manual of Clemson Automotive Training System (CATS)

1.1 The components of CATS

CATS is assembles with several parts (refer to picture 1). The computer hardware includes a Dell OptiPlex 980 desktop with an Intel i7 processor, a 1GB high-performance graphics card, and a 19-inch monitor equipped with a sound bar. It also includes a racing chair, which can move forwards and backwards to adjust the distance between the driver and pedals, a Logitech G27 steering wheel with 900 degrees of rotation, and a foot pedal assembly containing the throttle, brake, and clutch. The steering wheel, pedals, and a seat belt are mounted to the racing chair assembly.

1.2 Basic view of throttle, and brake pedal, and steering wheel

The steering wheel is placed in front of driver, and driver can put their hands on the steering wheel easily. During driving, turn the steering wheel left or right can control the vehicle directions. The range of this steering wheel is 900 degree. There are two pedals, which are located behind the steering wheel respectly. The vehicle in the CATS is Automotive Transmission, and it is set by moving forward and moving backward. There are three options in this gear: move backward, neutral gear (stopped), and move forward. The default gear is move forward, means the car is moving forward when starting the simulator. Click on the left pedal once, the gear will change to neutral gear, the vehicle will lose power, click on the left pedal twice, the gear will go to move backward mode. At the same time, when in the reverse mode, click on the right pedal twice can change the gear setting back to move forward.

The pedals, from left to right, they are clutch, brake, and gas pedals respectively. They are located right infront of the driver's feet. Adjusting the racing chair to the place where the driver can use these throttles comfortably. Since the vehicle is default set to automotive transmission, we do not need to use the clutch. Here we only need to use brake pedal, and gas pedal. The brake pedal is in the middle, it is used to decrease the speed of velocity. The brake pedal has 255 levels set by the sensor. When pressed in different force, the velocity will decrease its speed in different speed. The gas pedal is on the right. When the vehicle needs to speed up, press the gas pedal, the gas pedal has has 255 levels set by the sensor, Press in different force change the acceleration of the vehicle.

1.3 How to run CATS

CATS is running on Ubuntu, the Linux operating system. In order to run CATS, there are several simple ways to do.

Step 1: Open a terminal. Click on the icon, which is a black window at the top of the screen in the icon bar (refer to picture F.1).

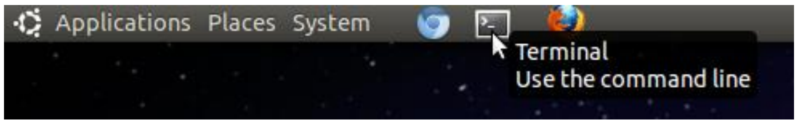
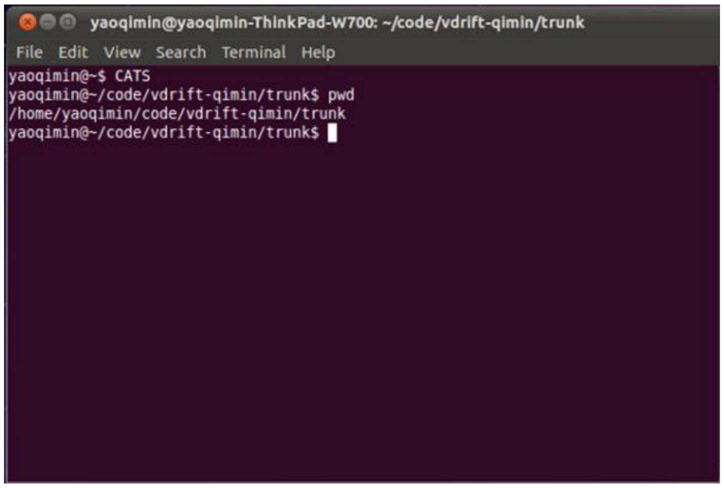


Figure F.1: View of opening terminal

Step 2: Go to the directory where CATS is; a shortcut was made for user. Simply type the command "CATS" in the terminal. Then go to the directory where CATS is (refer to picture F.2). The command also offered a function to check the directory by typing command: "pwd" (present working directory).



```
yaoqimin@yaoqimin-ThinkPad-W700: ~/code/vdrift-qimin/trunk
File  Edit  View  Search  Terminal  Help
yaoqimin@~$ CATS
yaoqimin@~/code/vdrift-qimin/trunk$ pwd
/home/yaoqimin/code/vdrift-qimin/trunk
yaoqimin@~/code/vdrift-qimin/trunk$ 
```

Figure F.2: Type command to CATS directory

Step 3: The developer offered another shortcut command, type "./1.sh" to run CATS; it is a bash file which implements a bunch of commands, so that when the CATS runs, it will process each step automatically. Don't worry how these functions work. A CATS menu will display on the screen (refer to picture F.3).

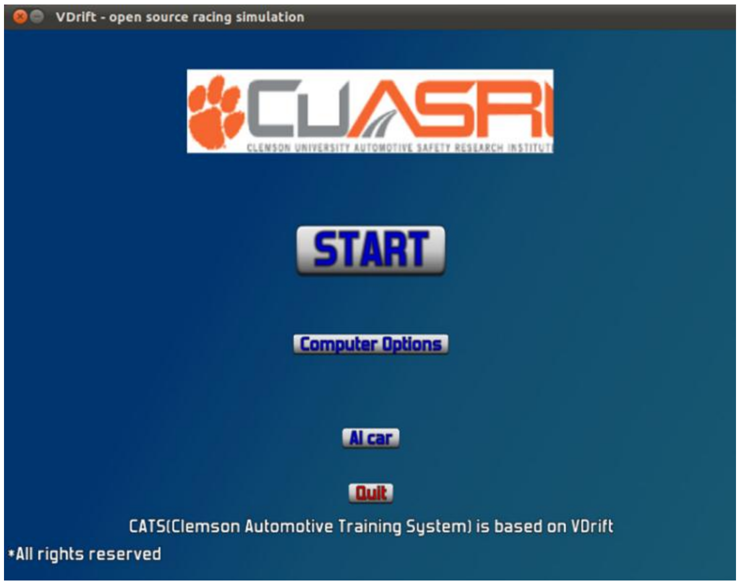

VDrift - open source racing simulation
CUASRI
CLEMSON UNIVERSITY AUTOMOTIVE SAFETY RESEARCH INSTITUTE
START
Computer Options
AI car
Quit
CATS(Clemson Automotive Training System) is based on VDrift
*All rights reserved

Figure F.3: Main menu of CATS

Step 4: In the CATS menu, there are four buttons, they are: Start, Computer Options, AI car, and Quit. In the computer option, you can change the settings for screen size, sound volumn, and font size. AI car is another function, which the driver can drive the vehicle, which interacts with another three artificial intelligence (AI) vehicles. Quit is used to quit the game. Since the computer option has already been set up by the developer, the driver only needs to click on the start button.

Step 5: In the CATS selection menu (refer to picture F.4), the driver can select the vehicle type on the top left. For example, the default vehicle is XS. A cube is located on the bottom left; click on a point in the cube to change the color of the vehicle. On the top right of the screen, two tracks are available for the user to choose: Practice track, which is used for drivers to become familiar with the pedals and the steering wheel. TigerPaw is another track for driving testing.

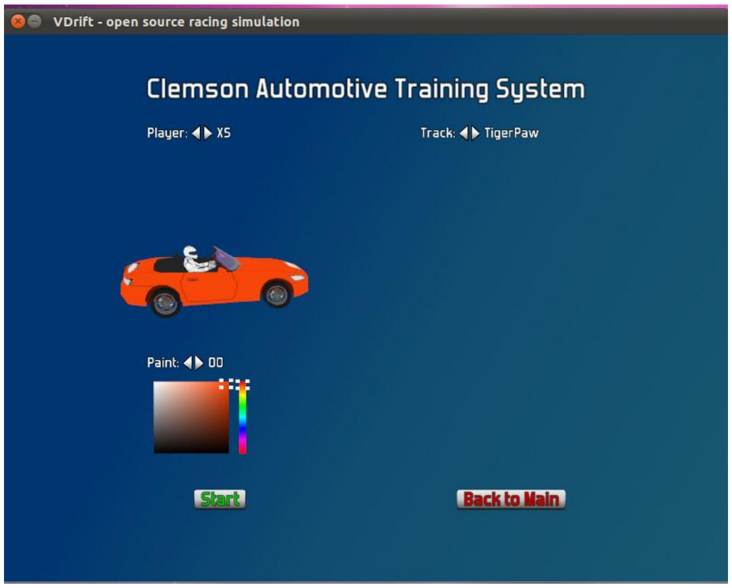

VDrift - open source racing simulation
Clemson Automotive Training System
Player: XS
Track: TigerPaw
Paint: 00
Start
Back to Main

Figure F.4: View of Track Selection Menu

Step 6: After the driver has clicked on the “Start” button, the simulator goes to the driving testing. Below is the basic view of driving simulator (refer to picture F.5). The vehicle is on the bottom middle. Press button “1”, “2”, or “3” to alter views. The default view is set for the driver to sit in the car and see the objects ahead of him, which is more reailistic compared to the other two views. On the bottom left, there are two bars: the bar on the top shows the velocity of the vehicle and the bar on the bottom displays the gas percentage and the gear number. On the top left of the screen, the number records the score during driving. A smile face displayed on the right side of the screen, which is one function offered by the feedback system. The illustration is covered in chapter 2.

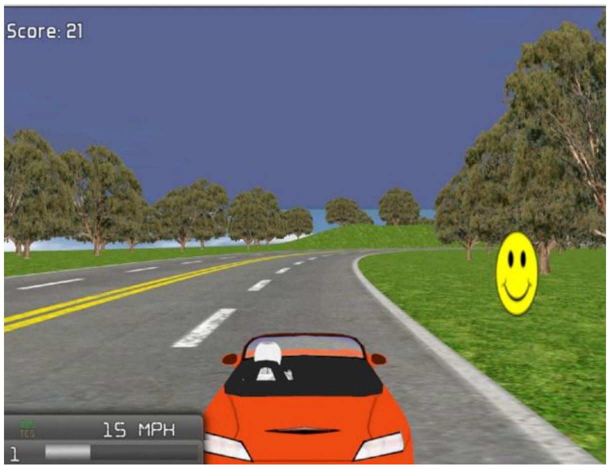


Figure F.5: View of virtural driving simulator

Step 7: Press “Esc”, and go to the pause menu. In the pause menu (refer to picture F.6), there are five options. “Resume” is to continue the driving simulator. “Restart Game” is to start the game again from the beginning. “Computer Option” is the same in the main menu; driver can adjust the screen size, sound volumn, font size, and buttons settings. However, the screen size will only be changed after restarting the CATS. The

"Leave" button is to quit the driving simulator and go back to the main manu. The "Score" button is to show the driver their specific driving scores based on each scenario.

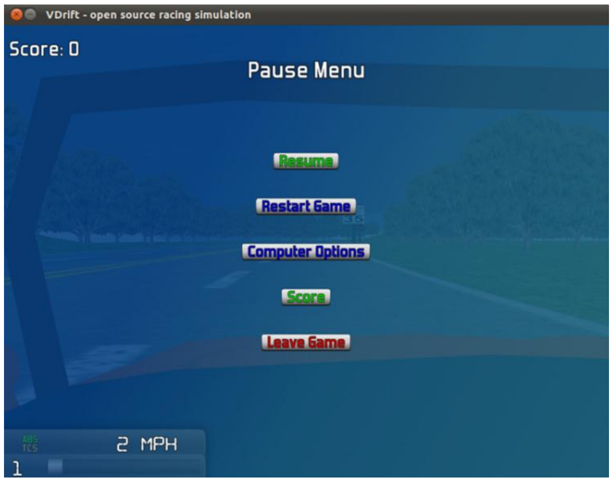
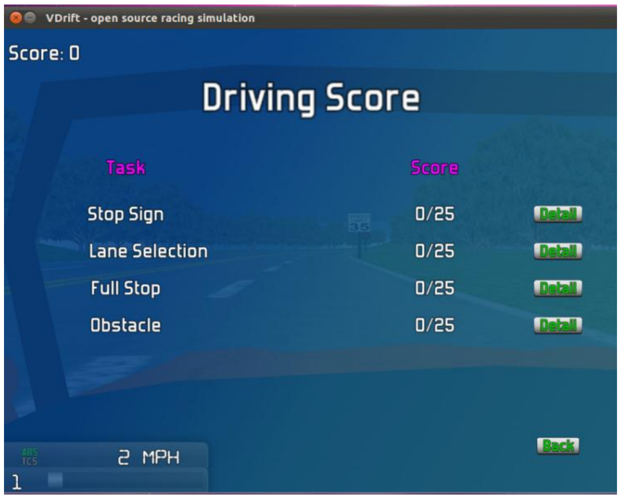


Figure F.6: View of pause menu

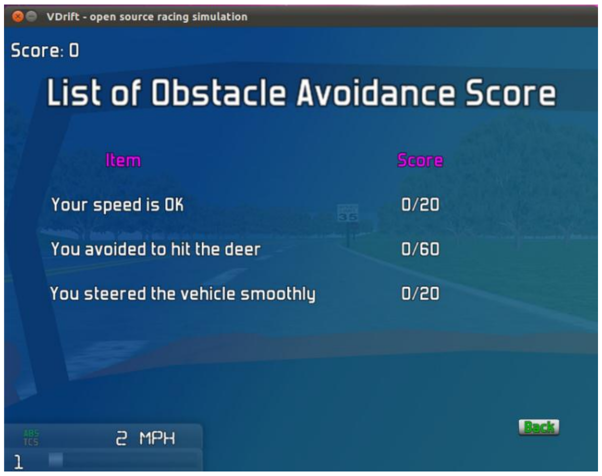
Step 8: After pressing the "Score" button, the Driving Score Menu (refer to picture F.7) appears. In this menu, it lists all the scenarios the driver will meet. "Task" shows the name of this scenario. "Score" lists the driving score out of the total score. On the right side of each score, there is a "Detail" button.



Task	Score
Stop Sign	0/25
Lane Selection	0/25
Full Stop	0/25
Obstacle	0/25

Figure F.7: View of driving score menu

Step 9: After pressing the "Detail" button, the driver can check his/her driving information. For example, the driver clicks on the "Detail" button of "Obstacle Avoidance Module" (refer to picture F.8). In this menu, it shows both the specific items the simulator detects and the score the driver actually got. Press the "back" button to leave this menu, and go to the previous menu.

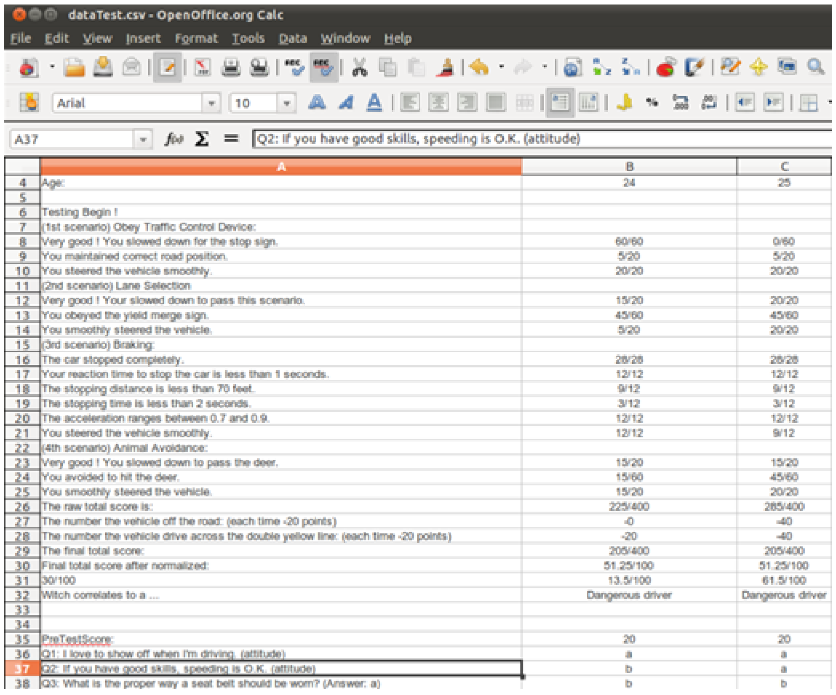


List of Obstacle Avoidance Score

Item	Score
Your speed is OK	0/20
You avoided to hit the deer	0/60
You steered the vehicle smoothly	0/20

Figure F.8: List of obstacle avoidance score

Step 10: After the driver has finished driving, they can check their driver information. A table (refer to picture F.9) records all the information the driver needs to read. For example, the score of each scenario, the score of pre and post-test questionnaire, and some general information such as the average velocity during the whole lap, sum of steering angle, are included. At the end of the driving score, a driving rating will be given for a driver's performance results.



dataTest.csv - OpenOffice.org Calc
File Edit View Insert Format Tools Data Window Help
Arial
10
A37

Q2: If you have good skills, speeding is O.K. (attitude)

	A	B	C
4	Age:	24	25
5			
6	Testing Begin !		
7	(1st scenario) Obey Traffic Control Device:		
8	Very good ! You slowed down for the stop sign.	60/60	0/60
9	You maintained correct road position.	5/20	5/20
10	You steered the vehicle smoothly.	20/20	20/20
11	(2nd scenario) Lane Selection		
12	Very good ! Your slowed down to pass this scenario.	15/20	20/20
13	You obeyed the yield merge sign.	45/60	45/60
14	You smoothly steered the vehicle.	5/20	20/20
15	(3rd scenario) Braking:		
16	The car stopped completely.	28/28	28/28
17	Your reaction time to stop the car is less than 1 seconds.	12/12	12/12
18	The stopping distance is less than 70 feet.	9/12	9/12
19	The stopping time is less than 2 seconds.	3/12	3/12
20	The acceleration ranges between 0.7 and 0.9.	12/12	12/12
21	You steered the vehicle smoothly.	12/12	9/12
22	(4th scenario) Animal Avoidance:		
23	Very good ! You slowed down to pass the deer.	15/20	15/20
24	You avoided to hit the deer.	15/60	45/60
25	You smoothly steered the vehicle.	15/20	20/20
26	The raw total score is:	225/400	285/400
27	The number the vehicle off the road: (each time -20 points)	-0	-40
28	The number the vehicle drive across the double yellow line: (each time -20 points)	-20	-40
29	The final total score:	205/400	205/400
30	Final total score after normalized:	51.25/100	51.25/100
31	30/100	13.5/100	61.5/100
32	Witch correlates to a ...	Dangerous driver	Dangerous driver
33			
34			
35	PreTestScore:	20	20
36	Q1: I love to show off when I'm driving. (attitude)	a	a
37	Q2: If you have good skills, speeding is O.K. (attitude)	b	a
38	Q3: What is the proper way a seat belt should be worn? (Answer: a)	b	b

Figure F.9: Driver’s information

Appendix G

Developer User Manual of Clemson Automotive Training System (CATS)

1.1 Note before installing VDrift:

Make sure the desktop/laptop to install VDrift is above the minimal requirement as followed,

a. Independent video card (NVIDIA is preferred) and 512MB or above

b. Steering wheel (we use G27 Logitech steering wheel) because all the settings in our customized VDrift are default to use this steering wheel

c. Core i5 or i7 is preferred because the calculation is huge in the simulator and high speed core can make sure the simulator runs well

Make sure the linux to install is the latest 32-bit Ubuntu version, the 64-bit Ubuntu version will have some problem issues when installing VDrift because some libraries do not match with the 64-bit Ubuntu version.

1.2 Download VDrift

1.3 Go to the VDrift official website www.vdrift.net

There are multiple choses to install VDrift, VDrfit can be installed in Windows, Mac, and Linux. In this particular situation, download the latest version of the Linux source codes.

1.3.2 After the VDrift package has been downloaded, uncompress the package through inputting the command: tar zxvf <package name> in the command terminal.

1.3.3 Since VDrift relies on a large number of library packages, before installing and compiling VDrift, all the library packages are required to be installed.

One easy way to install all the library packages is to input command:

Sudo apt-get install g++ scons libsdl-gfx1.2-dev libsdl-image1.2-dev libsdl-net1.2-dev libvorbis-dev libglew-dev libasio-dev libboost-dev

(Note: if the linux system installed is 64-bit, it has to install all the packages from the Ubuntu package center since most inputted packages do not support the 64-bit version)

1.3.4 In recently, people compile files through the command "make". Because of the large number of head files and cpp files, "scons" was used as default compile tool. The desktop in the lab is the latest and powerful one which has 4 cores, each core handles with 2 missions at the same time. For the "scons" tool, we can add one option "-j9" which can use full of the system resources to compile VDrift. The input command displays as belows:

```
Scons –j9
```

1.3.5 VDrift can be run after compiling the VDrift data. Input the command:

```
Build/vdrift
```

1.4 Basic Way to Create a Track

Today in the market, there are multiple ways to create a track. Some people use three-dimensional software such as 3D_MAX, and MAYA. But most people choose to build the track in some certain professional track builder software. Bob's Track Builder was used to model and build a professional track. Bob's Track Builder has several advantages. First of all, it is a mature track builder software in many aspects. It has powerful functions to help modify the height or the angle of the track's surface. Furthermore, it can add lots of objects around the track such as trees, vehicles, houses, and animals. Moreover, Bob's Track Builder can export the track to Blender in an easy way. Next some basic methods will be introduced to build a track.

Step1: Be familiar with the interface.

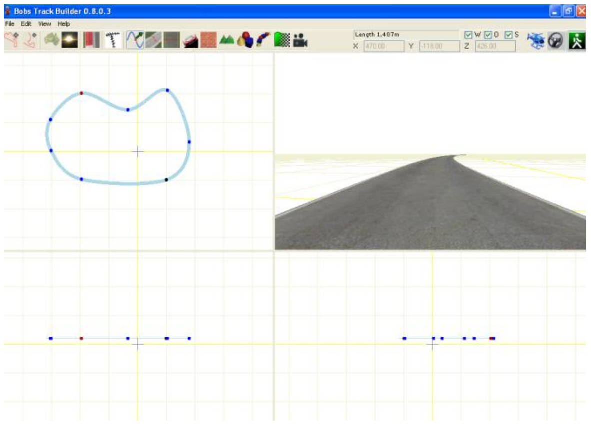


Figure G.1: Interface of Bob's Track Builder (BoB)

In figure G.1, it is the interface of the Bob's Track Builder, the slide bar on the top is the tool bar which we can use to add and edit most of the features. Then the main screen has been separated into 4 parts. The left top picture is the shape of the track. The right top picture is the track surface where we can edit the height, the slope and materials of the road. The left bottom picture is the front view of the road. The right bottom picture is the right view of the road.

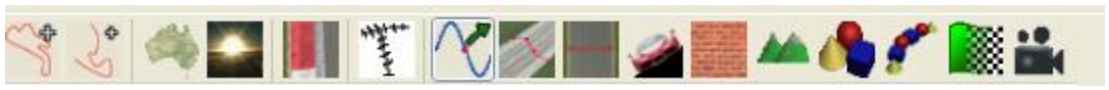


Figure G.2: Interface of tool Bar

Figure G.2 represents the tool bar, each picture represents a unique function as bellowed (order from left to right):

1. Create a closed loop track
2. Create an open track

3. Add a background image
4. Move the sunshine around
5. Edit materials
6. Edit/Switch active track
7. Move nodes and control points
8. Edit track surface properties
9. Edit the width of the track
10. Edit the camber of the track
11. Add walls
12. Edit surround terrain
13. Edit objects
14. Edit strings of objects
15. Edit start/finish position
16. Cameras

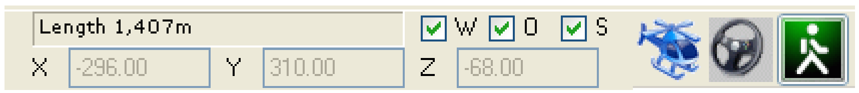

Length 1,407m
☑ W ☑ O ☑ S
X -296.00 Y 310.00 Z -68.00

Figure G.3: Interface of View Control

Figure G.3 represents an interface of view control which has three modes:

1. Helicopter: Go look the whole road in a high place
2. Steering wheel: Drive on the track
3. Person : Walk on the track

It is recommended to use the Person view to add and edit objects on the track.

Step 2: Create a track

Two options are available to create the track, one is to build a closed loop track, the other is to build a open loop track, since we need to add the start position and count the loop numbers in the later step, here the right way is to choose to build a closed loop track.

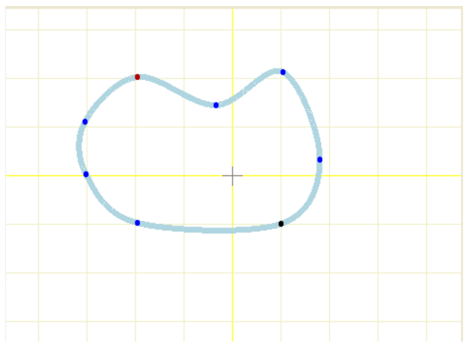
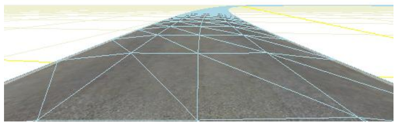


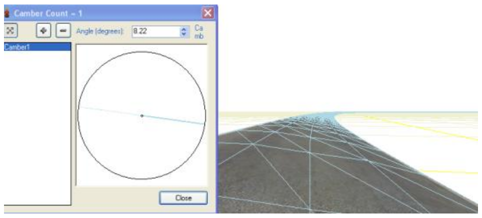
Figure G.4: Create a track

One thing need to notice is that, there are some nodes on the track, the more nodes are on the track, the better ability owned to control the shape of the track. On the other hand, when import the track to VDrift, there are many extra works to do. The developer recommends having the nodes no more than 15 nodes just for convenience.

Step 3: Edit the surface of the track

When the track is created, it's better to make the track as real as the daily driving road. In most time the road is not flat, some road has a bump and the other road has a small slope. Some small hills are created to add to the track. Bob's Track Builder gives a good opportunity to get all the features that are needed.





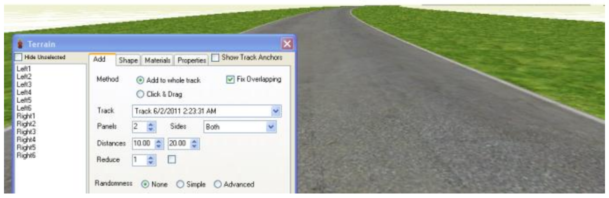
Camber Count = 1

Angle (degrees): 8.22

Figure G.5: Image of a road with a bump Figure G.6: Image of a road with slope

Step 4: Create a terrain

In the Figure G.5 and G.6, the track is not realistic, the main reason is that there is only a road with nothing else. The road lacks the other factors that combine the world together. Now it is better to have some grass along the road. Using the function to create a terrain as picture showed below.



Terrain

Hide Unselected: ☐

Show Track Anchors: ☐

Method: [✓ Add to whole track] Click & Drag

Fix Overlapping: ✓

Track: Track 6/2/2011 2:23:31 AM

Panels: 2 Sides: Both

Distances: 10.00 20.00

Reduce: 1 ☐

Randomness: [✓ None] Simple Advanced

Figure G.7: Grass on both sides of the road

This function is not only to create the grass; it can also control the width of the grass as needed. There are many parameters in this function, and they can used to change some different affects on the road. One another good feature this function has is that when we create a closed loop track, the grass can be added on both sides of the road. Then the big whole can be filled out with grass in the center of the closed loop track.

Step 5: Add String Objects on the Road

Sometimes people can see some curves on the road, these curves can also be obtained by using the function. Look at the picture

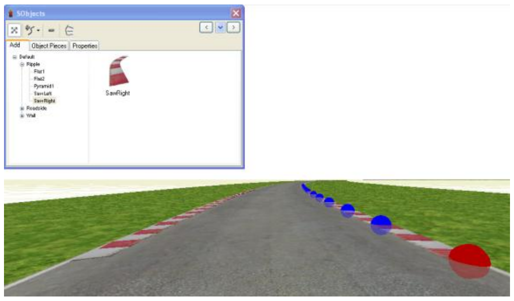
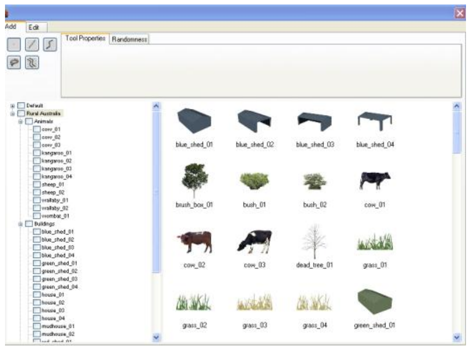


Figure G.8: Image of adding string objects on the road

It is a good practice to add some string objects along the road. By using the same method, we can also put some objects on the road.

Step 6: Add multiple objects

The Bob's Track Builder is one my favorite part, this program offers various objects including houses, animals, trees, etc. What we need to do is add all of the objects we like into the road like the figure G.9 shows



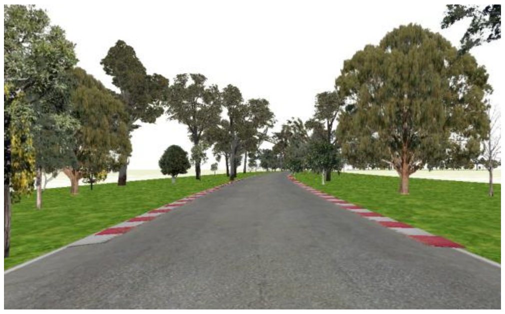


Figure G.9: Menu of multiple objects selections Figure G.10: Image of Trees on the road

Figure G.10 shows a good view of the trees on the road. There are three modes to add objects on the road and my personal preference is the 5th option, to add the objects on both sides of the road. When driving a car, it is not easy to hit the objects and be forced to stop. Besides the trees, many other objects such as animals and houses can be added on the road. See figure G.11

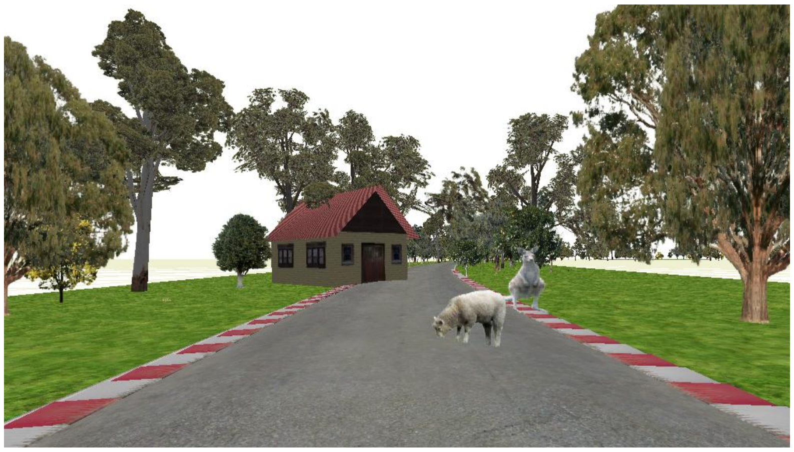


Figure G.11: Image of various objects on the road

Step 7: Output the data/track

This is one most important part for the whole track building. It needs to be very careful, since one simple track will usually has more than 50 objects, each objects accords to a certain data and its related files like photos. If unfornately a mistake is made, it’s very hard to find at the beginning and will become a disaster when editing the track in the Vdrift Track Editor.

Export the data twice in different format, the first data was export in DirectX format, which will be used in Blender to make the track and the objects. The second data to export in Racer format; it contains all the pictures of the objects that will be used in the track. In short, both of the data to export are very important.

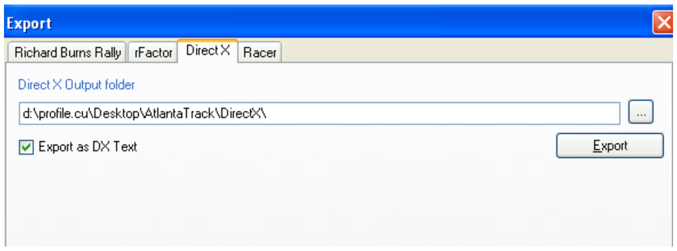
In conclusion, Bob’s Track Builder is a good track builder software that people can use during the daily life. It can do nearly everything needed for building a track. In addition, on the internet especially on the youtube and the official Bob’s Track Builder website: www.bobstrackbuilder.net you can find several training videos which I recommand will be very helpful.

1.5 Method to import Bob's Track Builder

Step 1: Export File DirectX and Racer from Bob's Track Builder to Blender

Bob's Track Builder is one commercial track building software that is popular among track builders. The advantage of this software is that it can build a track convenient and fast. In the previous chapter, the Bob's Track Builder has been introduced. Next is to discuss the method how to import the track made by Bob's Track Builder to Clemson Automotive Training System.

In this research project, there are several steps to export a track. First step is to export two files from Bob's Track Builder (refer to Figure G.12 and Figure G.13).


Export

Richard Burns Rally | rFactor | DirectX | Racer

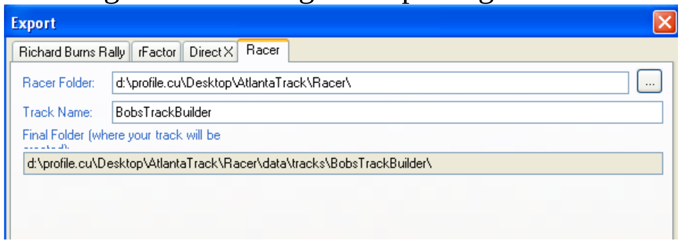
DirectX Output folder

d:\profile.cu\Desktop\AtlantaTrack\DirectX\

☑ Export as DX Text

Export

Figure G.12: Image of exporting DirectX


Export

Richard Burns Rally | rFactor | DirectX | Racer

Racer Folder: d:\profile.cu\Desktop\AtlantaTrack\Racer\

Track Name: BobsTrackBuilder

Final Folder (where your track will be created): d:\profile.cu\Desktop\AtlantaTrack\Racer\data\tracks\BobsTrackBuilder\

Figure G.13: Image of exporting Racer

Two files are used to export from Bob's Track Builder. One file called DirectX which records the coordinate information of all the objects on the track. The other file called Racer which contains all the images of the object surfaces. These images will be used later in the Blender.

Step 2: Import DirectX and Racer to Blender and edit the track in Blender

Blender is an open source three dimensional building software. It is mainly be used to design and edit the objects in 3-D. Below is a basic view of Blender

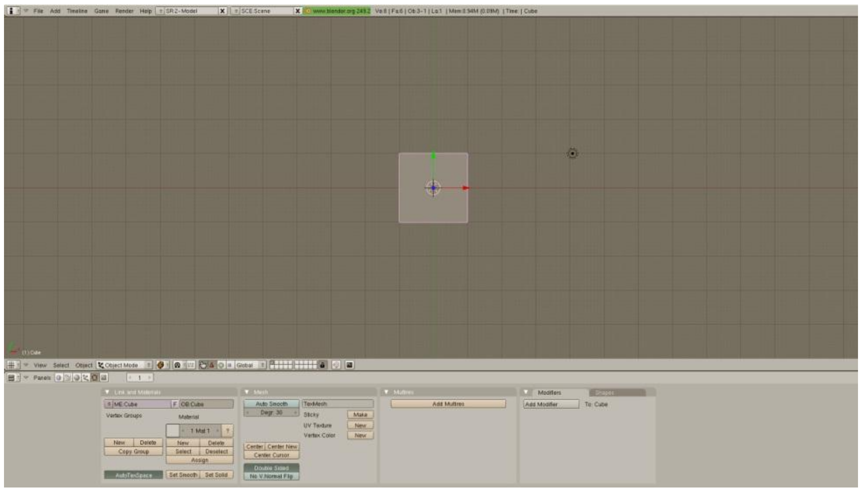


Figure G.14: Basic view of Blender

The user interface is simple, on the bottom of the image is the control, which we can use to deal with the object, at the center of the image is the place we make the objects, we can move and rotate the objects by clicking on this object. Right now there is no object in Blender, we need to import objects to Blender by importing files in DirectX. Click on the file button, select the import choice, then go to find the file we need then click on “ok”. G.15 give you some hints on how to do it.

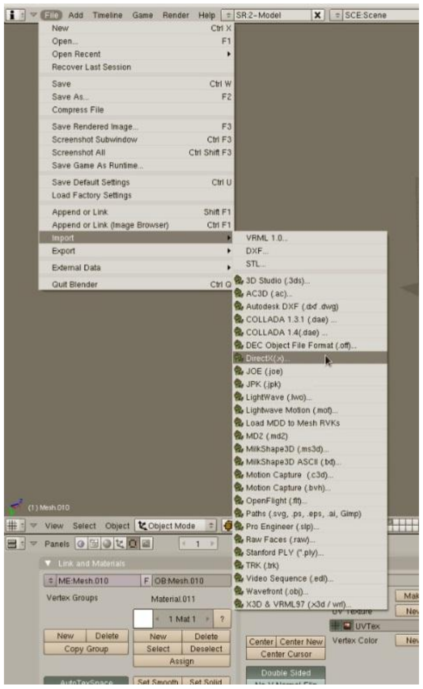


Figure G.15: Import of Importing DirectX to Blender

After the file has been imported into Blender, then we can see the object displayed on the screen.

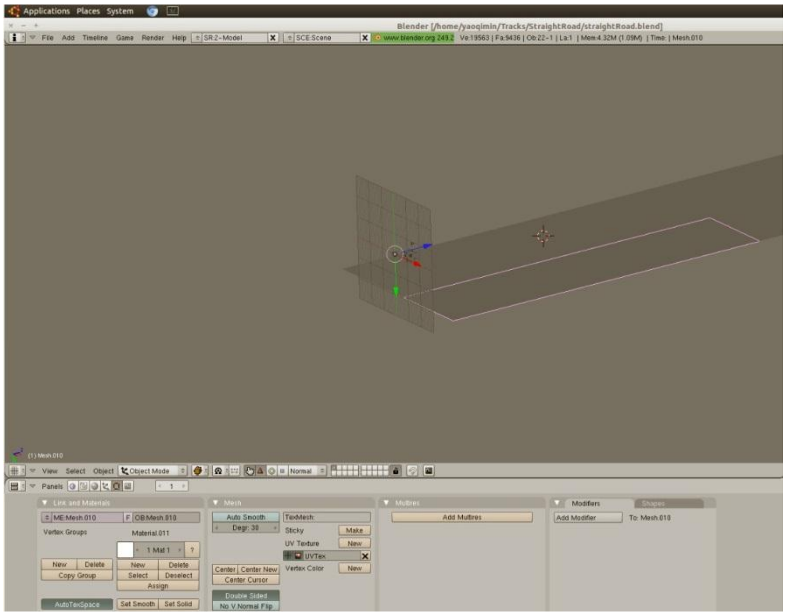


Figure G.16: Object displayed on the screen

We can move and rotate the object easily. Repeat the same steps for several times to import all the objects to the blender.

Next what we need to do is to relate the surface with its image. Before doing it, we need to do some configuration first. Right click on the border of control bar, image showed as below

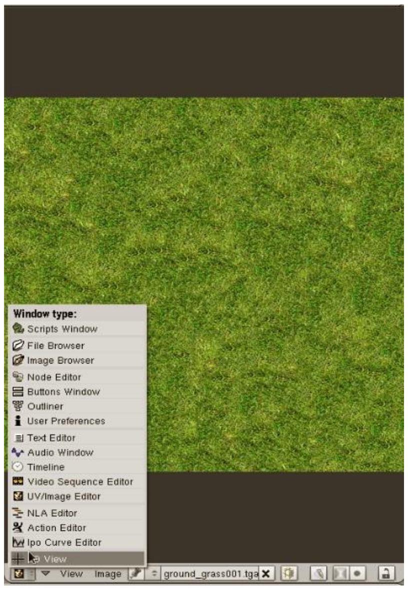


Figure G.17: Menu list in Blender

Select “UV image”, it is a mode we can attach the image onto the surface in Blender. For example, we can attach grass onto the surface. Repeat the same process until all the surfaces of our track has been attached an image. Based on my experience, it will be long and tedious, if you can find some easier way to attach the image, I will be glad to know.

Step 3: Export *.joe file to VDrift-TrackEditor

Since you have got all the staff you want, next step is to export your track to the VDrift-editor, a software that is written particularly for our CATS system. Click on file and select export choice showed as followed, select “joe”

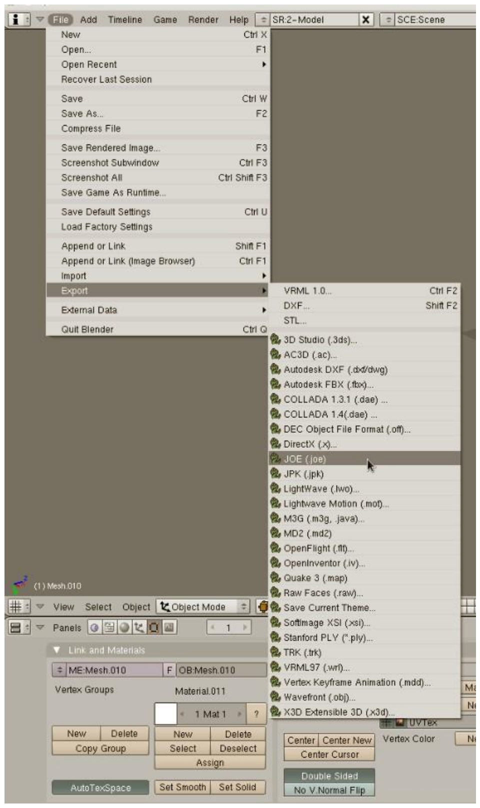


Figure G.18: Exporting track to track editor

This step is also tedious because there is no way to export the whole track at one time, the only way to do this is to export each part of the track step by step, the more objects in the track you have, the more time you need to spend on exporting the track. The path directory where all the joe files saved are required, save all the .joe files in /home/vdrift/vdrift-trackeditor/trunk/track/

1.6 How to create scenarios in the Clemson Automotive Training System

The driving simulator is developed based on VDrift which is written in c++. So before you start to develop this simulator by yourself, make sure that you have the basic fundamental of c++ programming skills. In order to help programmer to decrease the difficult for developing the simulator, some Python scripts have been used, Python is not required to be mastered. Following steps help you to start to create your own scenario on the track.

Step 1: Find the file, the default directory of our simulator is in

```
/home/vdrift/vdrift-qimin/trunk/
```

You can come to this file by typing:

```
cd ~/home/vdrift/vdrift-qimin/trunk/
```

Step 2: Create a new class which includes your new scenario.

In the CATS simulator, all the objects connect with each other, the programmer made a short cut for us. In the CATS directory, type:

```
cd tools/scripts/datametric-generator/
```

In the CATS directory, there is a place for people to create new scenario,

Step 3: Use Python to create new matrix, type in

```
Python create_new_matric “[The name of your scenario]”
```

For example:

```
Python create_new_matric “Stop Sign”
```

Step 4: Check whether the new class has been created.

The Class whose name is what you have typed has been created. Type:

```
cd  ~/home/vdrift/vdrift-qimin/trunk/src/
```

You can see “Datametric_stop_sign.cpp” in this directory. Congratulations! This is the first step.

Step 5: Add the new scenario to the list.

This step is to tell the simulator that you have created a new matric and you want this scenario to be run in the simulator, type:

```
cd ~/home/vdrift/.vdrift/data.config
```

Here you can see many files, look for the files that has the ”datametric”, add your new features following the same format and write the new list below the others, save and quit.

One thing need to be cautious, you are asked to input the input value and the output value, the input value means what kind of parameters you want to get in this scenarios, and the output values are the values you will calculate and they are the results. They must be listed in these columns, otherwise, the simulator will complain “Segmatation Fault” while driving.

For example:

Suppose I need to create the stop sign, and I want to know the velocity of the vehicle, the distance between the vehicle and the center of the track, whether the vehicle is off the road. I need to type:

```
Required_columns = Velocity, RunOffRoad, DistanceBetweenTrackandVehicle
```

Then I will give a score based on these parameters, then I need to type

```
Output_vars = Score
```

Step 6: Look at the class.

Here I attach part of the class code below. In the header file, we will see:

```
#ifndef STOPSIGNMETRIC_H                                    // line 1
#define STOPSIGNMETRIC_H
#include <vector>
#include <string>
#include "datamanager.h"

/* This is the place where you should describe your new DATAMETRIC-
  derived class. */
class STOPSIGNMETRIC : public DATAMETRIC
{
        public:                                             // line 11
           /* According to the Stop Sign Module, I add the same params */
          enum state_T {brake_now, braking, stopped};

                  /* Class constructor */
                STOPSIGNMETRIC(DATAMETRIC_CTOR_PARAMS_DEF);
                /* Class destructor */
                ~STOPSIGNMETRIC();
                /* Update the calculations, add to log, generate events */
                void Update(float dt);
        private:
                /*  Private member for Stop Sign Module */
                                                            // line 21
                        /*  Figure out which state it is now */
                void DetermineState();
                        /* Singleton object, only appear once during the game */
                static METRICTYPEREGISTER<STOPSIGNMETRIC> reg;
                /* Detect the distance between vehicle and stop sign */
                        double distanceScore;
                        /* Vehicle speed when it pass the stop sign */
                double speedScore;
                        /* Score for first scenario */
                double score1;                                      // line 31
                        /* Detect status of vehicle, such as driving, stopping , stopped
                state_T state;
                state_T last_state;
                        /* Get the time the vehicle use to stop */
                double stopping_time;
                        /* Get the speed when the vehicle is ready to stop */
                double final_speed;
                        /* Get the distance the vehicle use to stop */
                double stopping_distance;
                        /* Deceleration to stop the vehicle */                    // line 41
                double gs;

                        /* Get the position (x, y, z) of vehicle before stop */
                double pre_x;
                double pre_y;
                double pre_z;
                        /* Get the position (x, y, z) of vehicle when stopped */
                double post_x;
                double post_y;
                double post_z;
                        /* Get the game time before the vehicle stop */         // line 51
                double braking_start_time;
                        /* Get the game time when the vehicle has stopped */
                double braking_stop_time;
                        /* Get the velocity when the vehicle is ready to stop */
                double braking_start_velocity;
                double braking_stop_velocity;
                /* Create an array to record the game time */
                  std::vector<double> timeLog;
                  /* Create an array to record the vehile velocity during the first scenario */
                std::vector<double> velocityLog;                          // line 61
                /* Get the initial time of the game time */
                double initialTime;
                /* Create an array to record the throttle during the first scenario */
                std::vector<double> throttleLog;
                /* Create an array to record the brake data during the first scenario */
                  std::vector<double> brakeLog;
                std::vector<string> v;
                int speed_Score;
                /* Return a signal if the vehicle is driving off the road */
                bool driveOffTrack;                                       // line 71
                /* Return a signal if the vehicle is crossing double yellow line */
                bool driveAcrossLine;
                /* Get the score if the vehicle has not driven off  road or crossed double
yellow line */
                int position_score;
                /* Create an array to record the steering angle data during the first scenario */
                        std::vector<double> steerLog;
                /* Return the maximum steering angle */
                  double steerMax;
                  /* Sum the steering angle and return the number */
```

```
        double steerSum;                              // line 81
        /* Get a score of steering angle */
        int steer_score;
};

#endif // STOPSIGNMETRIC_H
```

In the code above, this is a typical head file, the name of this class is called: STOPSIGNMETRIC, it is inherited from DATAMETRIC. In the public member, there are two functions, they are constructor and destructor, they are used to allocate space for this new object. There are three members in this metric as an example, we use them to record the parameter we need.

Here is the code we use to implement the new features in STOPSIGNMETRIC:

```
#include <iostream>                                          // line 1
#include <assert.h>
#include <sstream>
#include <cmath>
#include <sstream>
#include <fstream>
#include <cstdlib>

using std::cout;
using std::endl;
using std::stringstream;
using std::ofstream;                                         // line 11
using std::ifstream;
using std::fstream;
using std::string;

#include "datametric_stop_sign.h"

/* Implement method for Stop Sign Module */
METRICTYPEREGISTER<STOPSIGNMETRIC>
STOPSIGNMETRIC::reg("StopSign");

/* Implement method of Stop Sign Module constructor */
STOPSIGNMETRIC::STOPSIGNMETRIC(DATAMETRIC_CTOR_PARAMS_DEF)
 : DATAMETRIC(DATAMETRIC_CTOR_PARAMS),
  /* All the data here is the same in the header file, initial all the value of parameters to
zero */                                                        // line 21
  distanceScore(0.0),
   speedScore(0.0),
   score1(0.0),
   state(brake_now),
   last_state(brake_now),
   stopping_time(0.0),
   final_speed(0.0),
   stopping_distance(0.0),
   gs(0.0),
   pre_x(0.0),                                                 // line 31
   pre_y(0.0),
   pre_z(0.0),
   post_x(0.0),
   post_y(0.0),
   post_z(0.0),
   braking_start_time(0.0),
   braking_stop_time(0.0),
   braking_start_velocity(0.0),
   braking_stop_velocity(0.0),
   timeLog(),                                                  // line 41
   velocityLog(),
   initialTime(0.0),
   throttleLog(),
   brakeLog(),
   speed_Score(0),
   driveOffTrack(false),
   driveAcrossLine(false),
   position_score(0),
   steerLog(),
   steerMax(0.0),                                         // line 51
   steerSum(0.0),
   steer_score(0)
//   detectCurrentSpeed(0.0),
//   detectLastSpeed(0.0)
{
        // Enabled by default
        DATAMETRIC::run = true;
}

/*  Implement methed of class destructor */
STOPSIGNMETRIC::~STOPSIGNMETRIC()                              // line 61
{
    /* Clean up any class data */
    timeLog.clear();
    velocityLog.clear();
}

/* Determine the state of the vehicle while driving */
void STOPSIGNMETRIC::DetermineState() {
  /* Detect vehicle state, if the state is brake_now, set the vehicle state to braking */
  if (state == brake_now) {
      if(DATAMETRIC::GetLastInColumn("Brake") > 0.0)                   // line 71
              state = braking;
   }
  /* Detect vehicle state, if the state is braking, set the vehicle state to stopped */
   else if (state == braking) {
      double velocity = DATAMETRIC::GetLastInColumn("Velocity");
           if(velocity < 3.0)
                  state = stopped;
         //  cout << "come to the braking state" << endl;
   }
  /* Detect vehicle state, if the state is stopped, set the vehicle state to its initial state,
brake_now */                                                           // line 81
   else if (state == stopped) {
      /* double velocity2 = DATAMETRIC::GetLastInColumn("Velocity");
           if(velocity2 > 20)
              state = brake_now; */
         // cout << "Come to the stopped state" << endl;
   }
}

/* Update all the data and call DetermineState function to change vehicle when necessary
*/
void STOPSIGNMETRIC::Update(float dt)
{                                                                  // line 91
         /* Do nothing when disabled */
         if (!run)
                  return;

         /*  Get existed velocity database from Excel */
         DATALOG::log_column_T const* velocity_col =
DATAMETRIC::GetColumn("Velocity");

         /* Get the most recent time */
         DATALOG::log_data_T velocity = velocity_col->back();
         /* Set the number of last digits of the velocity to two */
```

```
        //velocity = int((velocity + 0.05) * 10) / 10.0;
        /* Get the column of sector from database, Excel */                    //
line 101
        DATALOG::log_data_T sector = DATAMETRIC::GetLastInColumn("Sector");
        /* Get current sector number */
        DATALOG::log_data_T last_sector =
DATAMETRIC::GetNextLastInColumn("Sector");
        /* Get current lap number */
          DATALOG::log_data_T lap_number =
DATAMETRIC::GetLastInColumn("LapNumber");
          /* Get current distance between vehicle and center line of track */
          DATALOG::log_data_T distance =
          DATAMETRIC::GetLastInColumn("DistanceBetweenCarAndTrack");
          /* Get total score from database, here the total score is zero */
//      DATALOG::log_data_T last_score =
DATAMETRIC::GetLastInColumn("TotalScore");                          //line 111
        /* Define two stringstream type parameter to record data */
        stringstream ss;
        stringstream yy;
        /* Call function to update the vehicle state */
        DetermineState();
        /* Detect if the vehicle has driven close to the Stop Sign Scenario */
        if( (sector != last_sector) && (sector == 1))
        {
                if ((int(lap_number) + 1) % 2 == 0) {                       // line 121
                        /* Display a “stop sign” image to mention the driver, an intersection
ahead */

        DATAMETRIC::SetFeedbackImageEvent("stop_sign",4,0.2,2.0,1.0,0.2,0.8,0.7);
                        /* Start recording time for stop sign scenario */
                        initialTime = DATAMETRIC::GetLastInColumn("Time");
                        // timeLog.push_back(DATAMETRIC::GetLastInColumn("Time"));
                }
        }

        if( sector == 1) {
          /* Get current time, velocity , throttle , brake, and steering angle value */
                timeLog.push_back(DATAMETRIC::GetLastInColumn("Time"));
                                                                        // line 131
          velocityLog.push_back(DATAMETRIC::GetLastInColumn("Velocity"));
          throttleLog.push_back(DATAMETRIC::GetLastInColumn("Throttle"));
          brakeLog.push_back(DATAMETRIC::GetLastInColumn("Brake"));
          steerLog.push_back(DATAMETRIC::GetLastInColumn("Steering"));
        }
        /* Detect if the vehicle has driven passed the Stop Sign Scenario */
        if( (sector != last_sector) && (sector == 2))
        {
                if ((int(lap_number) + 1) % 2 == 0)
                {                                                       // line 141
                          /* Time Part */
                    stringstream timeStream;
                    /* Record time during this scenario and save them to
timeFor1stScenario.dat */
                    std::vector<DATALOG::log_data_T>::iterator it;
                        for(it = timeLog.begin(); it != timeLog.end(); ++it) {
                          timeStream << "," << (*it - initialTime);
                        }
                        /* Open file and save time data in this file */
                              ofstream
                     outFileLS("/home/vdrift/Desktop/1st/timeFor1stScenario.dat");
                        if(outFileLS.is_open()) {                        // line 151
                          outFileLS << timeStream.str();
                        }
                        /*  Close file */
                        outFileLS.close();
                                    /* Velocity Part */
                        stringstream velocityStream;
                        /* Record velocity during this scenario and save to
speedFor1stScenario.dat */
                        for(std::vector<double>::iterator itVelocity = velocityLog.begin();
                                itVelocity != velocityLog.end(); ++itVelocity) {
                          velocityStream << "," << *itVelocity;          // line 161
                        }
                        /* Open file and save data */
                        ofstream
outFileVelocity("/home/vdrift/Desktop/1st/speedFor1stScenario.dat");
                        if(outFileVelocity.is_open()) {
                          outFileVelocity << velocityStream.str();
                        }
                        /* Throttle Part */
                        stringstream throttleStream;
                              /* Record throttle during this scenario and save to
                     throttleFor1stScenario.dat */
                        for(std::vector<double>::iterator itThrottle = throttleLog.begin();
                        itThrottle != throttleLog.end(); ++itThrottle) {   // line 171

                          throttleStream << "," << *itThrottle;
                        }
                        /* Open file and save data */
                        ofstream
outFileThrottle("/home/vdrift/Desktop/1st/throttleFor1stScenario.dat");
                        if(outFileThrottle.is_open()) {
                          outFileThrottle << throttleStream.str();
                        }
                        /* Brake Part */
                        stringstream brakeStream;                      // line 181
                              /* Record brake during this scenario and save to
                     brakeFor1stScenario.dat */
                        for(std::vector<double>::iterator itBrake = brakeLog.begin();
                                itBrake != brakeLog.end(); ++itBrake) {
                          brakeStream << "," << *itBrake;
                        }
                        /* Open file and save data */
                        ofstream
outFileBrake("/home/vdrift/Desktop/1st/brakeFor1stScenario.dat");
                        if(outFileBrake.is_open()) {
                          outFileBrake << brakeStream.str();
                        }                                              // line 191

                        /* Normal Step */
                        /* Get speed when vehicle is passing through white line
                                    Under the stop sign */
                        final_speed = DATAMETRIC::GetLastInColumn("Velocity");
                        /* Keep two digits after number */
                              final_speed = int((final_speed + 0.05) * 10) / 10.0;
                              /* Print out the final speed */
                        cout << "The final speed for stop sign is: " << final_speed << endl;
                                    /* Words for 1st Scenario */              // line 201
                        /* If vehicle velocity is lower than 8 km/h, then give the score, 15
points */
                              if(velocity < 8)
                        {
                          speedScore = 15;
                          ss << "Have stopped before the stop sign" << endl;}
                        /* If vehicle velocity is higher than 8 km/h, then we regards
                                the vehicle has not stopped before the stop sign */
                              else
                        {
                          ss << "Have not stopped before the stop sign" << endl;// line 211
//                   last_score -= 20;
                        }
                        /* If the distance between vehicle and centerline of track
```

```
                        Is lower than 5 meters, it means the vehicle is on the track */
                    if(distance < 5)
                    {
                        /* Give distance score, 10 points */
                            distanceScore = 10;
                        ss << "Drive on the road" << endl;
                    }                                                   // line 221
                    else
                    {
                        /* If the vehicle is not driving on the track , give a
                                        Warning, the vehicle is not on the track */
                            ss << "Warning! not on the road" << endl;
                    }
                    /* Display an instance message on the screen */
//
        DATAMETRIC::SetFeedbackMessageEvent("stop_sign",ss.str(),2.0,1.0,0.2);
                        /* If the vehicle velocity is lower than 8 km/h,
                          display "Smile face" on the screen */                    // line 230
                        if(velocity < 8) {
                        DATAMETRIC::SetFeedbackImageEvent("smile_sign",6,0.2,2.0,
                        1.0,0.2,0.6,0.1);
                    }
                            /* If the vehicle velocity is between 8 km/h and 20 km/h,
display a "normal
                      Face" on the screen */
                    else if( velocity > 8 && velocity < 20) {

DATAMETRIC::SetFeedbackImageEvent("ok_sign",7,0.2,2.0,1.0,0.2,0.6,0.1);
                    }
                    Else                                                // line 241

          /* Display an "cry face" image on the screen */
          DATAMETRIC::SetFeedbackImageEvent("cry_sign",8,0.2,2.0,1.0,0.2,0.6,0.1);

        /* Check if the vehicle is driving off track */
         double tempOffTrack = DATAMETRIC::GetLastInColumn("offRoad");
         if(tempOffTrack == 4) driveOffTrack = true;
        /* Check if the vehicle is crossing double yellow line */
         double tempAcrossLine = DATAMETRIC::GetLastInColumn("LeftOrRight");
         if(tempAcrossLine == -1 ) driveAcrossLine = true;

            /* Score  been send to the dataTest.csv */
         ifstream inFile;                                               // line 251
        /* Open file and save data */
```

```
 inFile.open("dataTest.csv");
/ * Print if open file succeed */
 if(!inFile) {
   std::cout << "Open initialDataTest.csv" << std::endl;
   inFile.open("initialDataTest.csv");
 }
     /* Read file line by line and save all data to the vector v */
 while(!inFile.eof()) {
    string oneLine;                                         // line 261
    getline(inFile,oneLine);
    v.push_back(oneLine);
 }
      /* Close file */
 inFile.close();
     /* Change vector v size to 100 */
 v.resize(100);
     /* Strategy to get socre */

    /* Set Velocity Score */                                 // line 271
 /* Set score weight to indicate the importance of each item */
 /* Weight for the vehicle velocity */
   int weight1 = 15;
   /* Weight for driving off road */
 int weight2 = 5;
 /* Weight for driving across double yellow line */
   int weight3 = 5;
   /* If the vehicle velocity is lower than 5 km/h, the driver can get full points */
 if(final_speed <= 5)
   speed_Score = 4;                                         // line 281
 /* If the vehicle velocity is between 5 km/h and 10 km/h, the driver
    Can get partial points */
 else if(final_speed > 5 && final_speed <= 10)
        speed_Score = 3;

  /* If the vehicle velocity is between 5 km/h and 10 km/h, the driver
    Can get partial points */
else if(final_speed > 10 && final_speed <= 15)
        speed_Score = 2;                                  // line 291
 else if(final_speed > 15 && final_speed <= 20)
        speed_Score = 1;
/* No points will get if the driver has not stopped the vehicle */
  else
        speed_Score = 0;
/* Multiply the score with the weight to get the final score */
```

```
        speed_Score *= weight1;
        stringstream s_speed;
        s_speed << speed_Score << "/60;";
        /* Save the score data to vector */                    // line 301
        v[7] += s_speed.str();

        /* Detect if the vehicle is maintained on correct road position */
        /* Get full credits if the vehicle is keeping on the road */
         if(driveOffTrack && driveAcrossLine)
         position_score = 4;
        /* Get partial credits if the vehicle is two wheels off */
        else if(driveAcrossLine && !driveOffTrack)
              position_score = 3;
       /* Get partial credits if the vehicle is two wheels off */
        else if(!driveAcrossLine && driveOffTrack)                   // line 311
              position_score = 2;
       /* None or few credits will be got if vehicle is off track is close to off road */
        else
              position_score = 1;
       /* Multipley the position socre with the weight2 to get final score */
        position_score *= weight2;
        stringstream s_position;
        s_position << position_score << "/20;";
        v[8] += s_position.str();

        /* Check if the driver steered the vehicle smoothly */          // line 321
        for(std::vector<double>::iterator itSteer = steerLog.begin(); itSteer !=
steerLog.end(); ++itSteer)     {
          steerSum += *itSteer;
          if(steerMax < *itSteer) steerMax = *itSteer;
        }
        /* Get the absolution of the sum of steering angle */
        steerSum = abs(steerSum);
        /* If the maximum steering angle is lower than 90,
               and the sum of steering angle is lower than 1800, get full credits */
        if(steerSum <= 1800 && steerMax <= 90) steer_score = 4;        // line 331
        /* If the sum of steering angle is either higher than 1800 or
             The maximum steering angle is higher than 90, get partial credits */
        else if(steerSum <= 1800 && steerMax > 90) steer_score = 3;

       /* If the sum of steering angle is either higher than 1800 or
             The maximum steering angle is higher than 90, get partial credits */
         else if(steerSum > 1800 && steerMax <= 90) steer_score = 2;
       /* If the sum of steering angle is either higher than 1800 or
       The maximum steering angle is higher than 90, get partial credits */
 else steer_score = 1;
/* Multiply the steering score with the weight3 */                    // line 341
  steer_score *= weight3;
 /* Save steering angle data to vector */
  stringstream s_steer;
  s_steer << steer_score << "/20;";
  v[9] += s_steer.str();

  /* Save data and send to the dataTest.csv */
  std::vector<string>::iterator it11;
 /* Open file and save vector data to this file */
  ofstream outFile("dataTest.csv");
  if(outFile.is_open()) {                                       // line 351
    for(it11 = v.begin(); it11 != v.end(); ++it11) {
      outFile << *it11 << std::endl;
    }
  }
 /* Close file */
  outFile.close();
}
        else
        {                                                              // line 361
                /* Diagnose the system error by printing some messages in the terminal
*/
                        yy << "Stop before the Stop Sign" << endl;
                        /* Display message on the screen */

DATAMETRIC::SetFeedbackMessageEvent("SSmessage",yy.str(),2.0,1.0,0.2);
        }
}

    /* Detect the vehicle statement on first scenario */
/* Set a certain distance for dectecting */
if ( (last_state != state) && (sector == 1)) {
/* Check the vehicle status */                                    // line 371
  if(state == braking) {
       /* If the status is braking,  get current data including time, velocity,
       Vehicle three dimension coordinates, X, Y , Z, and record its braking time */
     braking_start_time = DATAMETRIC::GetLastInColumn("Time");
          braking_start_velocity = DATAMETRIC::GetLastInColumn("Velocity");
          pre_x = DATAMETRIC::GetLastInColumn("CarPositionX");
          pre_y = DATAMETRIC::GetLastInColumn("CarPositionY");
          pre_z = DATAMETRIC::GetLastInColumn("CarPositionZ");
```

```
            cout << "The start braking time is: " << braking_start_time << endl;
        }                                                   // line 381
        else if(state == stopped) {
      /* Check the vehicle status, if the status is stopped, the second time to get
      Time, braking_stop time, velocity, vehicle coordinates positions */
        braking_stop_time = DATAMETRIC::GetLastInColumn("Time");
            cout << "The stop braking time is: " << braking_stop_time << endl;
            braking_stop_velocity = DATAMETRIC::GetLastInColumn("Velocity");
            post_x = DATAMETRIC::GetLastInColumn("CarPositionX");
            post_y = DATAMETRIC::GetLastInColumn("CarPositionY");
            post_z = DATAMETRIC::GetLastInColumn("CarPositionZ");

          /* Get the stopping time by substract braking stop time to braking start time */

        stopping_time = braking_stop_time - braking_start_time;       // line 391
       /* Get last two digits number of stopping_time */
         stopping_time = int((stopping_time + 0.05) * 10) / 10.0;
           /* Calculate the stopping_disntance */
        stopping_distance = sqrt( (post_x - pre_x)*(post_x - pre_x) + (post_y -
pre_y)*(post_y - pre_y) + (post_z - pre_z)*(post_z - pre_z) );
       /* Convert from mph to km/h
        stopping_distance *= 2.2369;
        /* Get last two digits number of stopping_distance */
        stopping_distance = int((stopping_distance + 0.05) * 10) / 10.0; // line 401
        /* Calculate deceleration */
        gs = ( (braking_stop_velocity - braking_start_velocity) / stopping_time) / 9.81;
        /* Convert from mph to km/h */
          gs *= 2.2369;
          /* Get last two digits number of deceleration */
        gs = int((gs + 0.05) * 10) / 10.0;

          /* Test result for stopping_time, stopping_distance and gs */
        cout << "The stopping time for stop sign is: " << stopping_time << endl;
        cout << "The stopping distance for stop sign is: " << stopping_distance << endl;
        cout << "The acceleration for stop sign is: " << gs << endl;      // line 411
        }
      }
         /* Assign last_state */
      last_state = state;
          /* Assign to the score 1_x */
      double score1_1 = speedScore;
      double score1_2 = distanceScore;
      //score1 = speedScore + distanceScore;
         /* Sum score for each item in the stop sign scenario */
```

```
score1 = speed_Score + position_score + steer_score;
   /* Normalized score by dividing 4 */                          // line 421
score1 /= 4;
   /* Output all the score to the excel file */
//output_data["score1_1"] = score1_1;
output_data["score1_1"] = speed_Score;
//output_data["score1_2"] = score1_2;
output_data["score1_2"] = position_score;
output_data["score1_3"] = steer_score;
output_data["score1"] = score1;


   /* Output parameters to the total score module */
output_data["param1_1"] = stopping_time;                          // line 431
output_data["param1_2"] = stopping_distance;
output_data["param1_3"] = final_speed;
output_data["param1_4"] = gs;
```

In the entire body of codes that has been listed above, first we need to look at the constructor, this is the entrance of the whole class, all the parameters should be initialized in the constructor. Jump to the DATAMETROC::update(). We can use the parameters we made to calculate the result we want. There are several functions we can use in this class. DATAMETRIC::SetFeedbackMessageEvent() is used to display message on the screen. For example, when the car go through the stop sign, the instructions will be displayed on the screen. The message is controlled by the yy.str() in the code. You can also use other ways to display message. In addition, DATAEMTRIC::SetFeedbackImageEvent() is used to display image on the screen. It is the same way to use these two functions.

Step 8: Get value of vehicle dynamics

In our simulator, we have created some tools for you to use. It called the data-manager, you can get many different kind of parameters from this manger. Usually, it is not easy to explain every functions in detail, I will make an example aim to help you understand the basic develop ideas. Now I have a stop sign function, and I want to know, if the vehicle stopped in front of the stop sign. We will use two parameters, first is the vehicle velocity, second is the track sector, track sector is what we used to mark the objects we made on the track. Then we can type:

DATALOG::log_data_T v = DATAMETRIC::GetLastInColumn(“Vector”)

Using the same way, we can get the sector, then we can write a “if-statement” to find if the vehicle velocity is equal to zero, and the sector has changed to one at the same time.

We also have other functions to get different parameters such as:

```
DATALOG::log_column_T const* velocity_col =
DATAMETRIC::GetColumn("Velocity");
DATALOG::log_data_T v = velocity_col->back()
```

Either of them can be used in the class, the second method is recommended to be used because it not only can get the current value but it can trace back all the parameters as well. For example, it we want to know the average of the velocity, we can do as followed:

```
for(DATALOG::log_column_T::const_iterator iterVelocity = datacolVelocity-
>begin();iterVelocity != datacolVelocity->end(); ++iterVelocity)
    {
        if(maxVelocity < *iterVelocity)
          maxVelocity = *iterVelocity;
        DATALOG::log_data_T value = *iterVelocity;
        sum += value;
        count++;
    }
averageVelocity = (sum / count) * 2.2369;
```

We can use the iterator which is very handful in c++ to iterate the whole velocity and get the result we want.

Step 9: Output value to the background process

We are interested in getting some parameters such as yaw angle, maximum velocity and lateral acceleration. We want to send these parameters to MATLAB for further calculations and look for some potential risks that may make. We have several ways to send these parameters.

First, we can send these parameters to Excel:

```
output_data["score1_1"] = score1_1;
```

the output_data is a specific format that we will use to send the data

score1_1 is the parameter we want to send

Second, we can also send these parameters to the MATLAB

```
ofstream outFileTime("/home/yaoqimin/matlab/bin/time.dat");
if(outFileTime.is_open())
{
        outFileTime << timeStr.str();
}
outFileTime.close();
```

We use stream to collect the parameters, then we give it a directory and save the date to a specific file. After that, the MATLAB will read the date from the file can draw the figures.

Step 10: Display the feedback on the screen

After we get some results, we show tell the driver whether he did right or not. Here we have another to use, type:

```
DATAMETRIC::SetFeedbackMessageEvent("SSmessage",yy.str(),2.0,1.0,0.2);
```

First argument (“SSmessage”) : the name of this event

Second argument (yy.str) : a stream that contain the information that we want to display on the screen

Third argument (2.0) : the period of time the message start displaying on the screen after the event has been triggered

Fourth argument (1.0) : the period of time the message will keep displaying on the screen

Fifth argument (0.2) : the period of time the message fade out of the screen

DATAMETRIC::SetFeedbackImageEvent(("Speeding_up",3,0.2,2.0,1.0,0.2,0.7,0.7)

First argument (“Speeding_up”) : the name of the event

Second argument (3) : point at which image will be used

Third argument (0.2) : the period of time the message start displaying on the screen after the event has been triggered

Fourth argument (0.7) : the period of time the message will keep displaying on the screen

Fifth argument (0.7) : the period of time the message fade out of the screen

Appendix H

Sample of C++ for Clemson Automotive Training System

```
/* Section 1-    header file of  “Control Sign Module” */
#ifndef STOPSIGNMETRIC_H                                                    // line 1
#define STOPSIGNMETRIC_H
#include <vector>
#include <string>
#include "datamanager.h"

/* This is the place where you should describe your new DATAMETRIC-
  derived class. */
class STOPSIGNMETRIC : public DATAMETRIC
{
        public:                                                        // line 11
          /* According to the Stop Sign Module, I add the same params */
          enum state_T {brake_now, braking, stopped};

                  /* Class constructor */
                STOPSIGNMETRIC(DATAMETRIC_CTOR_PARAMS_DEF);
                /* Class destructor */
                ~STOPSIGNMETRIC();
                /* Update the calculations, add to log, generate events */
                void Update(float dt);
        private:
                /*  Private member for Stop Sign Module */                     // line 21
                    /*  Figure out which state it is now */
                void DetermineState();
                    /* Singleton object, only appear once during the game */
                static METRICTYPEREGISTER<STOPSIGNMETRIC> reg;
                /* Detect the distance between vehicle and stop sign */
                     double distanceScore;
                     /* Vehicle speed when it pass the stop sign */
                double speedScore;
                     /* Score for first scenario */
                double score1;                                             // line 31
                     /* Detect status of vehicle, such as driving, stopping , stopped
                state_T state;
                state_T last_state;
                     /* Get the time the vehicle use to stop */
                double stopping_time;
                     /* Get the speed when the vehicle is ready to stop */
                double final_speed;
                     /* Get the distance the vehicle use to stop */
                double stopping_distance;
                    /* Deceleration to stop the vehicle */                  // line 41
                double gs;

                    /* Get the position (x, y, z) of vehicle before stop */
                double pre_x;
                double pre_y;
                double pre_z;
                    /* Get the position (x, y, z) of vehicle when stopped */
                double post_x;
                double post_y;
                double post_z;
                    /* Get the game time before the vehicle stop */                    // line 51
                double braking_start_time;
                    /* Get the game time when the vehicle has stopped */
                double braking_stop_time;
                        /* Get the velocity when the vehicle is ready to stop */
                double braking_start_velocity;
                double braking_stop_velocity;
                /* Create an array to record the game time */
                    std::vector<double> timeLog;
                    /* Create an array to record the vehile velocity during the first scenario */
                std::vector<double> velocityLog;                                    // line 61
                /* Get the initial time of the game time */
                double initialTime;
                /* Create an array to record the throttle during the first scenario */
                std::vector<double> throttleLog;
                /* Create an array to record the brake data during the first scenario */
                    std::vector<double> brakeLog;
                std::vector<string> v;
                int speed_Score;
                /* Return a signal if the vehicle is driving off the road */
                bool driveOffTrack;                                             // line 71
                /* Return a signal if the vehicle is crossing double yellow line */
                bool driveAcrossLine;
                /* Get the score if the vehicle has not driven off  road or crossed double yellow line */
                int position_score;
                /* Create an array to record the steering angle data during the first scenario */
                        std::vector<double> steerLog;
                /* Return the maximum steering angle */
                    double steerMax;
                    /* Sum the steering angle and return the number */
                double steerSum;                                                    // line 81
                /* Get a score of steering angle */
                int steer_score;
};

#endif // STOPSIGNMETRIC_H
```

%Section 2 - Implement file of “Control Sign Module”

```
#include <iostream>                                             // line 1
#include <assert.h>
#include <sstream>
#include <cmath>
#include <sstream>
#include <fstream>
#include <cstdlib>

using std::cout;
using std::endl;
using std::stringstream;
using std::ofstream;                                            // line 11
using std::ifstream;
using std::fstream;
using std::string;

#include "datametric_stop_sign.h"

/* Implement method for Stop Sign Module */
METRICTYPEREGISTER<STOPSIGNMETRIC> STOPSIGNMETRIC::reg("StopSign");

/* Implement method of Stop Sign Module constructor */
STOPSIGNMETRIC::STOPSIGNMETRIC(DATAMETRIC_CTOR_PARAMS_DEF)
 : DATAMETRIC(DATAMETRIC_CTOR_PARAMS),
  /* All the data here is the same in the header file, initial all the value of parameters to zero */          // line
21
  distanceScore(0.0),
   speedScore(0.0),
   score1(0.0),
   state(brake_now),
   last_state(brake_now),
   stopping_time(0.0),
   final_speed(0.0),
   stopping_distance(0.0),
   gs(0.0),
   pre_x(0.0),                                                  // line 31
   pre_y(0.0),
   pre_z(0.0),
   post_x(0.0),
   post_y(0.0),
   post_z(0.0),
   braking_start_time(0.0),
   braking_stop_time(0.0),
   braking_start_velocity(0.0),
   braking_stop_velocity(0.0),
   timeLog(),                                                   // line 41
   velocityLog(),
   initialTime(0.0),
   throttleLog(),
   brakeLog(),
   speed_Score(0),
   driveOffTrack(false),
   driveAcrossLine(false),
   position_score(0),
   steerLog(),
   steerMax(0.0),                                                          // line 51
   steerSum(0.0),
   steer_score(0)
//   detectCurrentSpeed(0.0),
//   detectLastSpeed(0.0)
{
        // Enabled by default
        DATAMETRIC::run = true;
}

/*  Implement methed of class destructor */
STOPSIGNMETRIC::~STOPSIGNMETRIC()                                        // line 61
{
    /* Clean up any class data */
    timeLog.clear();
    velocityLog.clear();
}

/* Determine the state of the vehicle while driving */
void STOPSIGNMETRIC::DetermineState() {
  /* Detect vehicle state, if the state is brake_now, set the vehicle state to braking */
  if (state == brake_now) {
     if(DATAMETRIC::GetLastInColumn("Brake") > 0.0)                              // line 71
             state = braking;
   }
  /* Detect vehicle state, if the state is braking, set the vehicle state to stopped */
   else if (state == braking) {
     double velocity = DATAMETRIC::GetLastInColumn("Velocity");
          if(velocity < 3.0)
                 state = stopped;
        //  cout << "come to the braking state" << endl;
   }
  /* Detect vehicle state, if the state is stopped, set the vehicle state to its initial state, brake_now */    // line
81
   else if (state == stopped) {
     /* double velocity2 = DATAMETRIC::GetLastInColumn("Velocity");
          if(velocity2 > 20)
             state = brake_now; */
        // cout << "Come to the stopped state" << endl;
   }
}

/* Update all the data and call DetermineState function to change vehicle when necessary */
void STOPSIGNMETRIC::Update(float dt)
{                                                                       // line 91
        /* Do nothing when disabled */
        if (!run)
                return;
        /*  Get existed velocity database from Excel */
        DATALOG::log_column_T const* velocity_col = DATAMETRIC::GetColumn("Velocity");

        /* Get the most recent time */
        DATALOG::log_data_T velocity = velocity_col->back();
        /* Set the number of last digits of the velocity to two */
        //velocity = int((velocity + 0.05) * 10) / 10.0;
        /* Get the column of sector from database, Excel */                    // line 101
        DATALOG::log_data_T sector = DATAMETRIC::GetLastInColumn("Sector");
        /* Get current sector number */
        DATALOG::log_data_T last_sector = DATAMETRIC::GetNextLastInColumn("Sector");
        /* Get current lap number */
          DATALOG::log_data_T lap_number = DATAMETRIC::GetLastInColumn("LapNumber");
          /* Get current distance between vehicle and center line of track */
          DATALOG::log_data_T distance =
          DATAMETRIC::GetLastInColumn("DistanceBetweenCarAndTrack");
          /* Get total score from database, here the total score is zero */
//      DATALOG::log_data_T last_score = DATAMETRIC::GetLastInColumn("TotalScore");//line 111
        /* Define two stringstream type parameter to record data */
        stringstream ss;
        stringstream yy;
        /* Call function to update the vehicle state */
        DetermineState();
        /* Detect if the vehicle has driven close to the Stop Sign Scenario */
        if( (sector != last_sector) && (sector == 1))
        {
                if ((int(lap_number) + 1) % 2 == 0) {                           // line 121
                        /* Display a “stop sign” image to mention the driver, an intersection ahead */
                        DATAMETRIC::SetFeedbackImageEvent("stop_sign",4,0.2,2.0,1.0,0.2,0.8,0.7);
                        /* Start recording time for stop sign scenario */
                        initialTime = DATAMETRIC::GetLastInColumn("Time");
                        // timeLog.push_back(DATAMETRIC::GetLastInColumn("Time"));
                }
        }

        if( sector == 1) {
         /* Get current time, velocity , throttle , brake, and steering angle value */
         timeLog.push_back(DATAMETRIC::GetLastInColumn("Time"));                // line 131
         velocityLog.push_back(DATAMETRIC::GetLastInColumn("Velocity"));
         throttleLog.push_back(DATAMETRIC::GetLastInColumn("Throttle"));
         brakeLog.push_back(DATAMETRIC::GetLastInColumn("Brake"));
         steerLog.push_back(DATAMETRIC::GetLastInColumn("Steering"));
        }
        /* Detect if the vehicle has driven passed the Stop Sign Scenario */
        if( (sector != last_sector) && (sector == 2))
        {
                if ((int(lap_number) + 1) % 2 == 0)
                {                                                               // line 141
                      /* Time Part */
                   stringstream timeStream;
                   /* Record time during this scenario and save them to timeFor1stScenario.dat */
                   std::vector<DATALOG::log_data_T>::iterator it;
                        for(it = timeLog.begin(); it != timeLog.end(); ++it) {
```

```
  timeStream << "," << (*it - initialTime);
}
/* Open file and save time data in this file */
        ofstream outFileLS("/home/vdrift/Desktop/1st/timeFor1stScenario.dat");
if(outFileLS.is_open()) {                                          // line 151
  outFileLS << timeStream.str();
}
/*  Close file */
outFileLS.close();
        /* Velocity Part */
stringstream velocityStream;
/* Record velocity during this scenario and save to speedFor1stScenario.dat */
for(std::vector<double>::iterator itVelocity = velocityLog.begin();
           itVelocity != velocityLog.end(); ++itVelocity) {
   velocityStream << "," << *itVelocity;                           // line 161
}
/* Open file and save data */
ofstream outFileVelocity("/home/vdrift/Desktop/1st/speedFor1stScenario.dat");
if(outFileVelocity.is_open()) {
  outFileVelocity << velocityStream.str();
}
/* Throttle Part */
stringstream throttleStream;
        /* Record throttle during this scenario and save to throttleFor1stScenario.dat */
for(std::vector<double>::iterator itThrottle = throttleLog.begin();        // line 171
           itThrottle != throttleLog.end(); ++itThrottle) {
   throttleStream << "," << *itThrottle;
}
/* Open file and save data */
ofstream outFileThrottle("/home/vdrift/Desktop/1st/throttleFor1stScenario.dat");
if(outFileThrottle.is_open()) {
   outFileThrottle << throttleStream.str();
}
/* Brake Part */
stringstream brakeStream;                                          // line 181
        /* Record brake during this scenario and save to brakeFor1stScenario.dat */
for(std::vector<double>::iterator itBrake = brakeLog.begin();
           itBrake != brakeLog.end(); ++itBrake) {
   brakeStream << "," << *itBrake;
}
/* Open file and save data */
ofstream outFileBrake("/home/vdrift/Desktop/1st/brakeFor1stScenario.dat");
if(outFileBrake.is_open()) {
  outFileBrake << brakeStream.str();
}                                                              // line 191

/* Normal Step */
/* Get speed when vehicle is passing through white line
            Under the stop sign */
final_speed = DATAMETRIC::GetLastInColumn("Velocity");
/* Keep two digits after number */
        final_speed = int((final_speed + 0.05) * 10) / 10.0;
        /* Print out the final speed */
```

```
                    cout << "The final speed for stop sign is: " << final_speed << endl;
                         /* Words for 1st Scenario */                         // line 201
                    /* If vehicle velocity is lower than 8 km/h, then give the score, 15 points */
                         if(velocity < 8)
                    {
                     speedScore = 15;
                     ss << "Have stopped before the stop sign" << endl;}
                    /* If vehicle velocity is higher than 8 km/h, then we regards
                           the vehicle has not stopped before the stop sign */
                         else
                    {
                     ss << "Have not stopped before the stop sign" << endl;         // line 211
//                   last_score -= 20;
                    }
                    /* If the distance between vehicle and centerline of track
                      Is lower than 5 meters, it means the vehicle is on the track */
                    if(distance < 5)
                    {
                           /* Give distance score, 10 points */
                                  distanceScore = 10;
                           ss << "Drive on the road" << endl;
                    }                                                        // line 221
                    else
                    {
                           /* If the vehicle is not driving on the track , give a
                                    Warning, the vehicle is not on the track */
                                  ss << "Warning! not on the road" << endl;
                    }
                    /* Display an instance message on the screen */
//                  DATAMETRIC::SetFeedbackMessageEvent("stop_sign",ss.str(),2.0,1.0,0.2);
                         /* If the vehicle velocity is lower than 8 km/h,
                           display "Smile face" on the screen */                       // line 230
                         if(velocity < 8) {
                         DATAMETRIC::SetFeedbackImageEvent("smile_sign",6,0.2,2.0,1.0,0.2,0.6,0.1
                         );
                    }
                          /* If the vehicle velocity is between 8 km/h and 20 km/h, display a "normal
                      Face" on the screen */
                    else if( velocity > 8 && velocity < 20) {
                     DATAMETRIC::SetFeedbackImageEvent("ok_sign",7,0.2,2.0,1.0,0.2,0.6,0.1);
                    }
                    Else                                                     // line 241

          /* Display an "cry face" image on the screen */
          DATAMETRIC::SetFeedbackImageEvent("cry_sign",8,0.2,2.0,1.0,0.2,0.6,0.1);

        /* Check if the vehicle is driving off track */
         double tempOffTrack = DATAMETRIC::GetLastInColumn("offRoad");
         if(tempOffTrack == 4) driveOffTrack = true;
        /* Check if the vehicle is crossing double yellow line */
         double tempAcrossLine = DATAMETRIC::GetLastInColumn("LeftOrRight");
         if(tempAcrossLine == -1 ) driveAcrossLine = true;
```

```
   /* Score  been send to the dataTest.csv */
  ifstream inFile;
 /* Open file and save data */
  inFile.open("dataTest.csv");
 / * Print if open file succeed */
  if(!inFile) {
    std::cout << "Open initialDataTest.csv" << std::endl;
    inFile.open("initialDataTest.csv");
  }
    /* Read file line by line and save all data to the vector v */
  while(!inFile.eof()) {
     string oneLine;
     getline(inFile,oneLine);
     v.push_back(oneLine);
  }
     /* Close file */
  inFile.close();
    /* Change vector v size to 100 */
  v.resize(100);
    /* Strategy to get socre */

   /* Set Velocity Score */
  /* Set score weight to indicate the importance of each item */
  /* Weight for the vehicle velocity */
     int weight1 = 15;
     /* Weight for driving off road */
  int weight2 = 5;
  /* Weight for driving across double yellow line */
     int weight3 = 5;
     /* If the vehicle velocity is lower than 5 km/h, the driver can get full points */
  if(final_speed <= 5)
    speed_Score = 4;
  /* If the vehicle velocity is between 5 km/h and 10 km/h, the driver
     Can get partial points */
  else if(final_speed > 5 && final_speed <= 10)
          speed_Score = 3;

   /* If the vehicle velocity is between 5 km/h and 10 km/h, the driver
     Can get partial points */
 else if(final_speed > 10 && final_speed <= 15)
          speed_Score = 2;
  else if(final_speed > 15 && final_speed <= 20)
          speed_Score = 1;
/* No points will get if the driver has not stopped the vehicle */
    else
          speed_Score = 0;
 /* Multiply the score with the weight to get the final score */
  speed_Score *= weight1;
  stringstream s_speed;
  s_speed << speed_Score << "/60;";
  /* Save the score data to vector */
  v[7] += s_speed.str();
  /* Detect if the vehicle is maintained on correct road position */
  /* Get full credits if the vehicle is keeping on the road */
     if(driveOffTrack && driveAcrossLine)
    position_score = 4;
  /* Get partial credits if the vehicle is two wheels off */
  else if(driveAcrossLine && !driveOffTrack)
          position_score = 3;
/* Get partial credits if the vehicle is two wheels off */
   else if(!driveAcrossLine && driveOffTrack)                                   // line 311
          position_score = 2;
/* None or few credits will be got if vehicle is off track is close to off road */
   else
          position_score = 1;
 /* Multipley the position socre with the weight2 to get final score */
  position_score *= weight2;
  stringstream s_position;
  s_position << position_score << "/20;";
  v[8] += s_position.str();

  /* Check if the driver steered the vehicle smoothly */                      // line 321
  for(std::vector<double>::iterator itSteer = steerLog.begin(); itSteer != steerLog.end(); ++itSteer)
{
    steerSum += *itSteer;
    if(steerMax < *itSteer) steerMax = *itSteer;
  }
  /* Get the absolution of the sum of steering angle */
  steerSum = abs(steerSum);
  /* If the maximum steering angle is lower than 90,
       and the sum of steering angle is lower than 1800, get full credits */
  if(steerSum <= 1800 && steerMax <= 90) steer_score = 4;                       // line 331
  /* If the sum of steering angle is either higher than 1800 or
       The maximum steering angle is higher than 90, get partial credits */
  else if(steerSum <= 1800 && steerMax > 90) steer_score = 3;

/* If the sum of steering angle is either higher than 1800 or
       The maximum steering angle is higher than 90, get partial credits */
    else if(steerSum > 1800 && steerMax <= 90) steer_score = 2;
/* If the sum of steering angle is either higher than 1800 or
       The maximum steering angle is higher than 90, get partial credits */
 else steer_score = 1;
/* Multiply the steering score with the weight3 */                            // line 341
   steer_score *= weight3;
 /* Save steering angle data to vector */
  stringstream s_steer;
  s_steer << steer_score << "/20;";
  v[9] += s_steer.str();

  /* Save data and send to the dataTest.csv */
  std::vector<string>::iterator it11;
 /* Open file and save vector data to this file */
  ofstream outFile("dataTest.csv");
  if(outFile.is_open()) {                                                     // line 351
    for(it11 = v.begin(); it11 != v.end(); ++it11) {
       outFile << *it11 << std::endl;
     }
    }
   /* Close file */
    outFile.close();
  }
          else
          {                                                              // line 361
                   /* Diagnose the system error by printing some messages in the terminal */
                         yy << "Stop before the Stop Sign" << endl;
                         /* Display message on the screen */
                   DATAMETRIC::SetFeedbackMessageEvent("SSmessage",yy.str(),2.0,1.0,0.2);
          }
  }

    /* Detect the vehicle statement on first scenario */
  /* Set a certain distance for dectecting */
  if ( (last_state != state) && (sector == 1)) {
  /* Check the vehicle status */                                               // line 371
    if(state == braking) {
        /* If the status is braking,  get current data including time, velocity,
        Vehicle three dimension coordinates, X, Y , Z, and record its braking time */
      braking_start_time = DATAMETRIC::GetLastInColumn("Time");
             braking_start_velocity = DATAMETRIC::GetLastInColumn("Velocity");
             pre_x = DATAMETRIC::GetLastInColumn("CarPositionX");
             pre_y = DATAMETRIC::GetLastInColumn("CarPositionY");
             pre_z = DATAMETRIC::GetLastInColumn("CarPositionZ");
             cout << "The start braking time is: " << braking_start_time << endl;
    }                                                                        // line 381
    else if(state == stopped) {
  /* Check the vehicle status, if the status is stopped, the second time to get
  Time, braking_stop time, velocity, vehicle coordinates positions */
    braking_stop_time = DATAMETRIC::GetLastInColumn("Time");
             cout << "The stop braking time is: " << braking_stop_time << endl;
             braking_stop_velocity = DATAMETRIC::GetLastInColumn("Velocity");
             post_x = DATAMETRIC::GetLastInColumn("CarPositionX");
             post_y = DATAMETRIC::GetLastInColumn("CarPositionY");
             post_z = DATAMETRIC::GetLastInColumn("CarPositionZ");

      /* Get the stopping time by substract braking stop time to braking start time */          // line
391
    stopping_time = braking_stop_time - braking_start_time;
   /* Get last two digits number of stopping_time */
     stopping_time = int((stopping_time + 0.05) * 10) / 10.0;
      /* Calculate the stopping_disntance */
    stopping_distance = sqrt( (post_x - pre_x)*(post_x - pre_x) + (post_y - pre_y)*(post_y - pre_y) +
(post_z - pre_z)*(post_z - pre_z) );
   /* Convert from mph to km/h
    stopping_distance *= 2.2369;
    /* Get last two digits number of stopping_distance */
    stopping_distance = int((stopping_distance + 0.05) * 10) / 10.0;                // line 401
    /* Calculate deceleration */
    gs = ( (braking_stop_velocity - braking_start_velocity) / stopping_time) / 9.81;
```

```
          /* Convert from mph to km/h */
             gs *= 2.2369;
             /* Get last two digits number of deceleration */
          gs = int((gs + 0.05) * 10) / 10.0;

           /* Test result for stopping_time, stopping_distance and gs */
          cout << "The stopping time for stop sign is: " << stopping_time << endl;
          cout << "The stopping distance for stop sign is: " << stopping_distance << endl;
          cout << "The acceleration for stop sign is: " << gs << endl;                         // line 411
          }
       }
         /* Assign last_state */
       last_state = state;
          /* Assign to the score 1_x */
       double score1_1 = speedScore;
       double score1_2 = distanceScore;
       //score1 = speedScore + distanceScore;
         /* Sum score for each item in the stop sign scenario */
       score1 = speed_Score + position_score + steer_score;
         /* Normalized score by dividing 4 */                                         // line 421
       score1 /= 4;
         /* Output all the score to the excel file */
       //output_data["score1_1"] = score1_1;
       output_data["score1_1"] = speed_Score;
       //output_data["score1_2"] = score1_2;
       output_data["score1_2"] = position_score;
       output_data["score1_3"] = steer_score;
       output_data["score1"] = score1;


         /* Output parameters to the total score module */
       output_data["param1_1"] = stopping_time;                                     // line 431
       output_data["param1_2"] = stopping_distance;
       output_data["param1_3"] = final_speed;
       output_data["param1_4"] = gs;
}
```

%Section 3-   Header file of “Right Lane Selection Module”

```
/* Head file, including all the parameters that we gonna use in this class  */
#ifndef LEFTORRIGHTMETRIC_H
#define LEFTORRIGHTMETRIC_H

#include <vector>
#include <string>
#include "datamanager.h"
/* This is the place where you should describe your new DATAMETRIC- derived class. */
class LEFTORRIGHTMETRIC : public DATAMETRIC
{
        public:
                /* Constructor */
                LEFTORRIGHTMETRIC(DATAMETRIC_CTOR_PARAMS_DEF);
                /* Destructor */
                ~LEFTORRIGHTMETRIC();
                /** Update the calculations, add to log, generate events */
                void Update(float dt);
        protected:
        private:
                /* Singletion method, only appear once in the CATS */
                    static METRICTYPEREGISTER<LEFTORRIGHTMETRIC>
                /* Declear a score for lane selection */
                    double laneScore;
                /* Declear a vehicle velocity parameter , and get score*/
                    double speedScore;
                /* Declear a parameter to measure the distance between
                 vehicle and centerline of track, and get score */
                    double distanceScore;
                    /* Declear a parameter to measure yaw rate and get score */
                double yawRateScore;
                /* Declear score2 for total score in the second scenario */
                double score2;
                /* Declear a parameter to measure the steering wheel angle */
                double sumAngle;
                /* Declear a steering angle to measure the maximum value */
                double peakSteeringAngle;
                /* Declear a parameter to count the number of speeding */
                double countVelocity;
                /* Declear a parameter to calculate the average velocity */
                double averageVelocity;
                /* Declear a parameter to sum the whole velocity */
                double sumVelocity;
                /* Declear a parameter to calculate the maximum velocity */
                double peakVelocity;
                /* Declear a parameter to get braking value */
                double brake;
                /* Declear a parameter to get throttle value */
                double throttle;

                /* Declear a parameter to get the vehicle position */
                    double lateralPosition;
                /* Declear a parameter to get the score based on the velocity */
```

```
            int score_velocity;                                          // line 51
            /* Declear an array to store a bouch of string */
        std::vector<string> v;
        /* Declear a parameter to check if the vehicle is driving off road */
        bool offRoad;
        /* Declear a parameter to check if the vehicle is left or right of the lane */
        bool rightLane;
        /* Declear a parameter to record the score based on driving off road factor */
        int score_offRoad;
        /* Declar a parameter to get score based on steering angle */
            int score_steer;                                             // line 61
            /* Create an array to store time */
        std::vector<double> timeLog;
        /* Create an array to store velocity */
        std::vector<double> velocityLog;
        /* Create an array to store throttle */
        std::vector<double> throttleLog;
        /* Create an array to store throttle */
        std::vector<double> brakeLog;
        /* Declear and set an initial time */
        double initialTime;                                          // line 71

};

#endif // LEFTORRIGHTMETRIC_H
```

%Section 4-   Implement file of “Right Lane Selection Module”

```
/* Implement method and funtions of Right Lane Selection Module */                              // line 1
#include <iostream>
#include <assert.h>
#include <sstream>
#include <cmath>
#include <cstdlib>
#include <fstream>

using std::cout;
using std::endl;
using std::stringstream;
using std::ofstream;                                                                    // line 11
using std::ifstream;
using std::fstream;
using std::string;

#include "datametric_left_or_right.h"

/// static class member declaration, register metric type name
METRICTYPEREGISTER<LEFTORRIGHTMETRIC> LEFTORRIGHTMETRIC::reg("LeftOrRight");
/* Convert constructor */
LEFTORRIGHTMETRIC::LEFTORRIGHTMETRIC(DATAMETRIC_CTOR_PARAMS_DEF)
 : DATAMETRIC(DATAMETRIC_CTOR_PARAMS) ,
/* Allocate space and assign initial value to all parameters declared in the header file */                    // line
21
  laneScore(0.0),
  speedScore(0.0),
  distanceScore(0.0),
  yawRateScore(0.0),
  score2(0.0),
  sumAngle(0.0),
  peakSteeringAngle(0.0),
  countVelocity(0.0),
  averageVelocity(0.0),
  sumVelocity(0.0),                                                                     // line 31
  peakVelocity(0.0),
  brake(0.0),
  throttle(0.0),
  lateralPosition(0.0),
  score_velocity(0),
  v(),
  offRoad(false),
  rightLane(false),
  score_offRoad(0),
  score_steer(0),                                                                             // line 41
  timeLog(),
  velocityLog(),
  throttleLog(),
  brakeLog(),
  initialTime(0.0)
{
        // Enabled by default
        DATAMETRIC::run = true;

}
/* Implementation of destructor */                                              // line 51
LEFTORRIGHTMETRIC::~LEFTORRIGHTMETRIC()
{
        // Clean up any class data
}
/* Implementation of update functions, call run function */
void LEFTORRIGHTMETRIC::Update(float dt)
{
        // Do nothing when disabled
        if (!run)
                return;                                                         // line 61

  /* Collect data of steering angle for period of time */
   //std::vector<double> sumSteeringAngle;

        /* Get current sector */
        DATALOG::log_data_T sector = DATAMETRIC::GetLastInColumn("Sector");
        /* Get the last second sector */
        DATALOG::log_data_T last_sector = DATAMETRIC::GetNextLastInColumn("Sector");
        /* Detect whether the vehicle is on left or right of track */
          DATALOG::log_data_T value = DATAMETRIC::GetLastInColumn("LeftOrRight");
        /* Get the number of the lap */                                         // line 71
          DATALOG::log_data_T lap_number = DATAMETRIC::GetLastInColumn("LapNumber");
        /* Get the value of distance between car and centerline of track */
          DATALOG::log_data_T distance =
DATAMETRIC::GetLastInColumn("DistanceBetweenCarAndTrack");
          /* Get velocity column */
        DATALOG::log_column_T const* velocity_col = DATAMETRIC::GetColumn("Velocity");
        /* Get current velocity */
          DATALOG::log_data_T velocity = velocity_col->back();
        /* Get current yaw rate */                                              // line 81
          DATALOG::log_data_T yaw_rate = DATAMETRIC::GetLastInColumn("YawRate");
        stringstream ss;
        stringstream yy;
/* Check if the vehicle is close to the Lane Selection Scenario */
if( (sector != last_sector) && (sector == 3))
{
        if((int(lap_number) + 1) % 2 == 0) {
                /* If the vehicle is close to the Lane Selection Scenario, display an “arrow_right”
                   Image on the screen */                                       // line 91
                DATAMETRIC::SetFeedbackImageEvent("arrow_right",2,0.5,2.0,1.0,0.2,0.8,0.0);
                /* At this point, get the current time, initial this current time to the initial time */
                    initialTime = DATAMETRIC::GetLastInColumn("Time");
        }
}

 /* Collect data during the whole sector 3 */
if(sector == 3) {
  /* Get sum of Steering Angle */
  sumAngle += abs(DATAMETRIC::GetLastInColumn("Steering"));
  /* Get Peak value of Steering Angle */                                    //line101
  if( peakSteeringAngle < DATAMETRIC::GetLastInColumn("Steering")) {
    peakSteeringAngle = DATAMETRIC::GetLastInColumn("Steering");
  }
  peakSteeringAngle = int((peakSteeringAngle + 0.05)*10) / 10.0;
  /* Get Sum of Velocity */
  sumVelocity += DATAMETRIC::GetLastInColumn("Velocity");
  countVelocity++;
  /* Get Peak of Velocity */
  if( peakVelocity < DATAMETRIC::GetLastInColumn("Velocity")) {
    peakVelocity = DATAMETRIC::GetLastInColumn("Velocity");                      //line111
  }
  peakVelocity = int((peakVelocity + 0.05)*10) / 10.0;
  /* Check if the vehicle is driving off the road */
  if(DATAMETRIC::GetLastInColumn("offRoad") != 4) offRoad = true;
  /* Check if the vehicle is driving on the correct lane */
  if(DATAMETRIC::GetLastInColumn("LeftOrRight") == -1) rightLane = true;

  /* Add time  to the vector */
  timeLog.push_back(DATAMETRIC::GetLastInColumn("Time"));
  /* Add velocity to the vector */
  velocityLog.push_back(DATAMETRIC::GetLastInColumn("Velocity"));                //line121
  /* Add throttle to the vector */
  throttleLog.push_back(DATAMETRIC::GetLastInColumn("Throttle"));
  /* Add brake to the vector */
  brakeLog.push_back(DATAMETRIC::GetLastInColumn("Brake"));
}

if( (sector != last_sector) && f(sector == 4))
{
/* Calcuate sum of steering angle for period of time here */
   if((int(lap_number) + 1) % 2 == 0)
         {                                                                  //line131
              /* Get Average of Velocity */
             averageVelocity = sumVelocity / countVelocity;
                   averageVelocity = int((averageVelocity + 0.05)*10) / 10.0;
             /* Get Throttle and Brake */
             brake = DATAMETRIC::GetLastInColumn("Brake");
                   throttle = DATAMETRIC::GetLastInColumn("Throttle");
                   brake = int((brake + 0.05)*10) / 10.0;
                   throttle = int((throttle + 0.05)*10) / 10.0;

                   /* Get Lateral Position */                                 //line141
                   lateralPosition = DATAMETRIC::GetLastInColumn("DistanceBetweenCarAndTrack");

                   /* Leave two digit numbers of lateralPosition */
                        lateralPosition = int((lateralPosition + 0.05)*10) / 10.0;
                   /* Detect if the vehicle is on the road */
                        if(value == 1)
                   {
                           /* Send wrong message to stringstream */
                                 ss << "Wrong Lane" << endl;
                   }
```

```
        else if(value == -1){                                          //line151
                /* Send right lane message to the stringstream */
                ss << "Correct Lane" << endl;
                /* Assign 10 points to the lane score */
                laneScore = 10.0;
        }
        /* Check the vehicle velocity */
        /* Under the condition if the vehicle velocity is lower than 40 */
            if(velocity < 40)
        {
                /* Assign the speed score as 5.0 */                        //line161
                      speedScore = 5.0;
                      /* Send message to string stream */
                ss << "Speed OK" << endl;
        }
        else
        {
                /* Send speeding message to string stream */
                      ss << "Speeding!" << endl;
        }
        /* Check if the vehicle is on the track */                         //line171
        if(distance < 5 && distance > 1)
        {
                /* Get full credits score */
                      distanceScore = 5.0;
                      /* Send message to the string stream */
                ss << "Drive on the correct lane" << endl;
        }
        else
        {
                /* Send warning message to the string stream */            //line181
                      ss << "Warning! not on the road" << endl;
        }
        /* Check if the absolution of yaw rate is lower than 50 */
        if(abs(yaw_rate) < 50)
        {
                yawRateScore = 5.0;
                /* Send message to string stream */
                ss << "Drive smoothly" << endl;
        }
        Else                                                           //line191
        {
                /* Send warning message to string stream */
                      ss << "Drive in a danger way" << endl;
        }

/* Send data to dataTest.csv */
/* Create File */
  ifstream inFile;
        /* Open file “”dataTest.csv” */
            inFile.open("dataTest.csv");

        if(!inFile) {                                                  //line201
```

```
       /* Message to confirm if the file has opened or created */
        std::cout << "Open initialDataTest.csv" << std::endl;
        inFile.open("initialDataTest.csv");
      }
      /* Read the file line by line until the end of this file */
      while(!inFile.eof()) {
        string oneLine;
        getline(inFile, oneLine);
       /* Save each line as an abstraction data to the vector */
        v.push_back(oneLine);                                    //line211
      }
      /* Close the file */
      inFile.close();
      /* Resize the vector to 100 */
      v.resize(100);

 /* Calculate score based on velocity */
int weight1 = 5;
int weight2 = 15;
int weight3 = 5;
/* If the maximum velocity is lower than 20 km/h, get full credits */          //line221
if(peakVelocity <= 20) score_velocity = 4;
/* If the maximum velocity is between 20 and 25 km/h, get partial credits */
else if(peakVelocity > 20 && peakVelocity <= 25) score_velocity = 3;
/* If the maximum velocity is between 20 and 25 km/h, get partial credits */
  else if(peakVelocity > 25 && peakVelocity <= 30) score_velocity = 2;
/* If the maximum velocity is between 20 and 25 km/h, get partial credits */
else if(peakVelocity > 30 && peakVelocity <= 50) score_velocity = 1;
/* No credits got if the maximum velocity is too high */
  else score_velocity = 0;
 /* Multiply the score velocity with weight1                                  //line231
score_velocity *= weight1;
stringstream s_speed;
  /* Print score_veloicty to the terminal for checking */
s_speed << score_velocity << "/20;";
  /* Assign score velocity to the vector */
v[11] += s_speed.str();


 /* Get full credits if the driver obeyed the yield merge sign */
 if(!offRoad && rightLane) score_offRoad = 4;
/* Get partial credits if the driver not obey both rules */
else if(offRoad && rightLane) score_offRoad = 3;                             //line241
/* Get partial credits if the driver not obey both rules */
  else if(!offRoad && !rightLane) score_offRoad = 2;
/* Get 1 credits if the driver did not obey both rules */
  else score_offRoad = 1;
  /* Multiply the score with weight2 */
score_offRoad *= weight2;
stringstream s_position;
  /* Print out final score to the terminal */
s_position << score_offRoad << "/60;";
/* Save position score to the vector */                                  //line251
```

```
     v[12] += s_position.str();

     /* Get full credits if driver steered the vehicle smoothly */
   if(sumAngle <= 1800 && peakSteeringAngle <= 90) score_steer = 4;
   /* Get partial credits if driver steered the vehicle not quite smoothly */
     else if(sumAngle <= 1800 && peakSteeringAngle > 90) score_steer = 3;
   /* Get partial credits if driver steered the vehicle not quite smoothly */
     else if(sumAngle > 1800 && peakSteeringAngle <= 90) score_steer = 2;
   /* Get one credit if not driving smoothly */
     else score_steer = 1;
   /* Multiply the steering angle score with weight3 */                              //line261
   score_steer *= weight3;
   stringstream s_steer;
   /* Save steering angle score to the s_steer vector */
   s_steer << score_steer << "/20;";
   v[13] += s_steer.str();

     /* Calculate the total score for the three items */
   double tempScore = score_velocity + score_offRoad + score_steer;
   //std::cout << "the temp score is: " << tempScore << std::endl;
   /* If the score is over 90, display smile face on the screen */
     if(tempScore >= 90)                                                       //line271
           DATAMETRIC::SetFeedbackImageEvent("smile_sign",6,0.2,2.0,1.0,0.2,0.6,0.1);
   /* If the score is between 70 and 90, display a normal face */
     else if(tempScore >= 70 && tempScore < 90)
           DATAMETRIC::SetFeedbackImageEvent("ok_sign",7,0.2,2.0,1.0,0.2,0.6,0.1); // 0.8 0.7
   /* If the score is lower than 70, display a cry face */
     else
           DATAMETRIC::SetFeedbackImageEvent("cry_sign",8,0.2,2.0,1.0,0.2,0.6,0.1);


      /* Save data and send to the dataTest.csv */
   std::vector<string>::iterator it;
   /* Open file */                                                             //line281
   ofstream outFile("dataTest.csv");
   if(outFile.is_open()) {
  /* Iterator vector from beginning to the end */
    for(it = v.begin(); it != v.end(); ++it) {
            outFile << *it << std::endl;
          }
   }
  /* Close file */
   outFile.close();

}                                                                          //line291
else
{
          /* Print out some instructional message */
          yy << "Please drive on the right lane" << endl;
          yy << "Follow the direction" << endl;
          /* Display message on the screen */
               DATAMETRIC::SetFeedbackMessageEvent("LRmessage",yy.str(),2.0,1.0,0.2);
}
```

```
	/* Send time data to MATLAB */
	stringstream timeStream;							//line301
	/* Iterator time vector from beginning to the end */
	std::vector<double>::iterator itTime;
	for(itTime = timeLog.begin(); itTime != timeLog.end(); ++itTime) {
	  timeStream << "," << (*itTime - initialTime);
	}
	   /* open file timeFor2ndScenario.dat */
	ofstream outFileTime("/home/vdrift/Desktop/2nd/timeFor2ndScenario.dat");
	/* Save time stream to the file */
	   if(outFileTime.is_open()) {
	  outFileTime << timeStream.str();						//line311
	}
	/* Close file */
	outFileTime.close();
	/* Send velocity data to MATLAB */
	stringstream velocityStream;
	/* Iterate velocity vector from beginning to the end */
	   for(std::vector<double>::iterator itVelocity = velocityLog.begin(); itVelocity !=
	   velocityLog.end(); ++itVelocity) {
	  velocityStream << "," << *itVelocity;
	}										//line321
	/* Open file */
	ofstream outFileVelocity("/home/vdrift/Desktop/2nd/velocityFor2ndScenario.dat");
	/* Save velocity data to the file */
	   if(outFileVelocity.is_open()) {
	  outFileVelocity << velocityStream.str();
	}
	/* Close file */
	outFileVelocity.close();


	   /* Send throttle data to MATLAB */
	stringstream throttleStream;							//line331
	/* Iterator throttle */
	for(std::vector<double>::iterator itThrottle = throttleLog.begin(); itThrottle != throttleLog.end();
++itThrottle) {
	  throttleStream << "," << *itThrottle;
	  //std::cout << *itThrottle << std::endl;
	}
	/* Open file and save data */
	ofstream outFileThrottle("/home/vdrift/Desktop/2nd/throttleFor2ndScenario.dat");
	if(outFileThrottle.is_open()) {
	  outFileThrottle << throttleStream.str();					//line341
	}
	/*  Close file */
	outFileThrottle.close();
	   /* Set brake parameter */
	stringstream brakeStream;
	for(std::vector<double>::iterator itBrake = brakeLog.begin(); itBrake != brakeLog.end(); ++itBrake)
{
	  brakeStream << "," << *itBrake;
```

```
        }
        /* Open file and save data */                                          //line351
        ofstream outFileBrake("/home/vdrift/Desktop/2nd/brakeFor2ndScenario.dat");
        if(outFileBrake.is_open()) {
          outFileBrake << brakeStream.str();
        }
        /* Close file */
        outFileBrake.close();
}
        /* Assign lane score to score2_1 */
        //double score2_1 = laneScore;
        /* Assign speed score to score2_2 */                                          //line361
        //double score2_2 = speedScore;
        /* Assign distance score to score2_3 */
        double score2_3 = distanceScore;
        /* Assign yawRateScore to score2_4 */
        //double score2_4 = yawRateScore;
        /* Sum score of each item to a total score */
//      score2 = laneScore + speedScore + distanceScore + yawRateScore;
        score2 = score_velocity + score_offRoad + score_steer;
        /* Normalize the total score */
          score2 /= 4;                                                          //line371
          /* Send score of each item back to the excel file */
        output_data["score2_2"] = score_velocity;
        output_data["score2_3"] = score_offRoad;
        output_data["score2_1"] = score2_3;
        output_data["score2_4"] = score_steer;
        output_data["score2"] = score2;
        output_data["param2_1"] = peakVelocity;
        output_data["param2_2"] = averageVelocity;
        output_data["param2_3"] = sumAngle;
        output_data["param2_4"] = peakSteeringAngle;                                  //line381
        output_data["param2_5"] = throttle;
        output_data["param2_6"] = brake;
        output_data["param2_7"] = lateralPosition;
}
```

%Section 5-   Header file of “Full Stop Module”

```
#ifndef FULLSTOPMETRIC_H                                                    //line1
#define FULLSTOPMETRIC_H
#include <vector>
#include "datamanager.h"
/** This is the place where you should describe your new DATAMETRIC-
 * derived class.
 */
class FULLSTOPMETRIC : public DATAMETRIC
{
        public:
                enum state_T {not_braking, brake_now, braking, stopped};             //line11
                /* Constructor */
                FULLSTOPMETRIC(DATAMETRIC_CTOR_PARAMS_DEF);
                /* Destructor */
                ~FULLSTOPMETRIC();
                /* Update the calculations, add to log, generate events */
                void Update(float dt);
        protected:
        private:
                /* Figure out what state the metric is in by Brake & Velocity values */
                void DetermineState();                                      //line21
                /* Current car state waiting for braking */
                state_T state;
                /* The state from the last update (to check for state changes) */
                    state_T last_state;
                    /* The velocity of the car when it began to brake */
                double braking_start_velocity;
                /* The time at which the braking stimulus was given */
                double braking_stimulus_time;
                /* The time at which braking began */
                double braking_start_time;                              //line31
                /* Object registering this type for creation by name */
                static METRICTYPEREGISTER<FULLSTOPMETRIC> reg; //!< object registering this
type for creation by name
                /* Whether the car has stopped in this scenario */
                double score3_1;
                /* The recreation time is les than 2 seconds */
                double score3_2;
                /* The stopping distance is less than 100 */
                    double score3_3;
                    /* The stopping time is lower than 5 secons */              //line41
                double score3_4;
                /* Total score for the 3rd scenario */
                double score3;
                /* Add a Boolean flag to detect the return value */
                bool kkflag;
                /* Add a Boolean flag to detect the return value */
                    bool kflag2; // add on March.7
                /* Global coordinates of vehicle before entering the scenario */
                    double pre_x;
                double pre_y;                                           //line51
                double pre_z;
                /* Global coordinates of vehicle after the scenario */
                double post_x;
                double post_y;
                double post_z;
                /* The time between the vehicle begin to stop and stop completely */
                double stopping_time;
                /* Decelaration of a vehicle */
                double acceleration;                                   //line61
                /* The time between image displayed on the screen and drive begin to press braking */
                double reaction_time;
                /* Distance between car began to stop and car stopped completely
                double distance;
                /* Boolean flag to detect velocity's status */
                bool kflag_velocity;
                /* Declear a vector velocity to store vector during the scenario */
                std::vector<double> velocityLog;
                /* Declear a time vector to store time during the scenario */
                std::vector<double> timeLog;                               //line71
                /* Record the minimal value of velocity */
                double min_velocity;
                /* Record the start time in the virtual driving simulator */
                double initial_time;
                /* Declear a throttle vector to store throttle during the scenario */
                std::vector<double> throttleLog;
                /* Declear a brake vector to store brake value during the scenario */
                    std::vector<double> brakeLog;
                    /* Delcear a vector called v */
                std::vector<string> v;                                     //line81
                /* Score to record if the vehicle has stopped */
                    int score_checkStopped;
                /* Score to record the time between driver saw the message and began to stop vehicle */
                    int score_reaction_time;
                /* Declear a score to record stopping distance
                    int score_stopping_distance;
                    /* Declear a score to record stopping time */
                int score_stopping_time;
                /* Declear a score to record deceleration */
                int score_acceleration;                                    //line91
                /* Record Steering angle value */
                std::vector<double> steer;
                /* Record the sum of steering angle */
                double sumSteer;
                /* Record the maximum of steering angle */
                double maxSteer;
                /* Record the score for steering angle */
                    int score_steer;
}; #endif // FULLSTOPMETRIC_H                                      //line101
```

%Section 6-   Implement file of “Full Stop Module”

```
#include <iostream>                                                                    //line1
#include <sstream>
#include <cmath>
#include <assert.h>
#include <cstdio>
#include <cstdlib>
#include <fstream>

using std::ostringstream;
using std::cout;
using std::endl;
using std::stringstream;                                                               //line11
using std::ofstream;
using std::ifstream;
using std::string;
using std::fstream;

#include "datametric_full_stop.h"
/// static class member declaration, register metric type name
METRICTYPEREGISTER<FULLSTOPMETRIC> FULLSTOPMETRIC::reg("FullStop");

/* Constructor and Assign initial value to all the decleared parameters */
FULLSTOPMETRIC::FULLSTOPMETRIC(DATAMETRIC_CTOR_PARAMS_DEF)
 : DATAMETRIC(DATAMETRIC_CTOR_PARAMS),                                                 //line21
   state(not_braking),
   //state(stopped),
   last_state(stopped),
   braking_start_time(-1.0),
   score3_1(0.0), score3_2(0.0), score3_3(0.0), score3_4(0.0),
   score3(0.0),
   kkflag(false),
   kflag2(true),
   pre_x(0.0),
   pre_y(0.0),                                                                         //line31
   pre_z(0.0),
   post_x(0.0),
   post_y(0.0),
   post_z(0.0),
   stopping_time(0.0),
   acceleration(0.0),
   reaction_time(0.0),
   distance(0.0),
   kflag_velocity(false),
   velocityLog(),                                                                      //line41
   timeLog(),
   min_velocity(0.0),
   initial_time(0.0),
   throttleLog(),
   brakeLog(),
   v(),
   score_checkStopped(0),
   score_reaction_time(0),
   score_stopping_distance(0),
   score_stopping_time(0),                                          //line51
   score_acceleration(0),
   steer(),
   sumSteer(0.0),
   maxSteer(0.0),
   score_steer(0)
{
        // Enabled by default
        DATAMETRIC::run = true;

}
/* Destructor */                                                   //line61
FULLSTOPMETRIC::~FULLSTOPMETRIC()
{
        // Clean up any class data
}

/* Function call to determine the state of vehicle during the scenario */
void FULLSTOPMETRIC::DetermineState()
{
        /* If the vehicle state is not_braking */
        if (state == not_braking)
        {                                                          //line71
                /* Check its speed, if the speed is over 55 mph */
                    if (DATAMETRIC::GetLastInColumn("Velocity") * 2.2369 > 55.0)
                        /* Change vehicle state */
                              state = brake_now;
        }
        /* If the vehicle state is brake_now */
        if (state == brake_now)
        {
                /* Get the current speed and check if it is bigger than 0 */
                    if (DATAMETRIC::GetLastInColumn("Brake") > 0.0)              //line81
                        /* Change vehicle state to braking */
                              state = braking;
        }
        /* If the vehicle state is braking now */
        else if (state == braking)
        {
                /* Get the vehicle velocity */
                    double velocity = DATAMETRIC::GetLastInColumn("Velocity");
                /* If the vehicle velocity is very small */
                    if (velocity <= 0.5)                                 //line91
                        /* Change the vehicle state to stopped  */
                        state = stopped;
                /* If the vehicle velocity is bigger than the previous vehicle velocity */
                    else if (velocity > DATAMETRIC::GetNextLastInColumn("Velocity"))
                        /* Change the vehicle state to not braking */
                        state = not_braking;
        }
        /* If the state is stopped */
        else if (state == stopped)
        {                                                                      //line101
                /* Check the vehicle velocity and set to the related vehicle state */
                    if (DATAMETRIC::GetLastInColumn("Velocity") > 0.5)
                {
                        /* Check brake, then set the state */
                              if (DATAMETRIC::GetLastInColumn("Brake") > 0.0)
                        {
                                state = braking;
                        }
                        else
                        {                                                      //line111
                                state = not_braking;
                        }
                }
        }
}
/* Call update function */
void FULLSTOPMETRIC::Update(float dt)
{

        //static kkflag = false;
        // Do nothing when disabled                                            //line121
        if (!run)
                return;
        /* Once the car has stopped, the kkflag will change to TRUE, so the full stop
          scenario will not be started again */
        if(kkflag)
                return;
        /* Get current lap sector */
          DATALOG::log_data_T sector = DATAMETRIC::GetLastInColumn("Sector");
        /* Get last lap sector */                                                      //line131
          DATALOG::log_data_T last_sector = DATAMETRIC::GetNextLastInColumn("Sector");
        /* Get lap number */
          DATALOG::log_data_T lap_number = DATAMETRIC::GetLastInColumn("LapNumber");
        /*Determine the vehicle state */
        DetermineState();


        /* Braking module is in sector 5 and sector 6, here it checks the braking module */
        if( sector == 5 || sector == 6) {
                /* Save current velocity to the vector velocity */
                    velocityLog.push_back(DATAMETRIC::GetLastInColumn("Velocity"));

                    /* Save current time to the time vector */                             //line141

                    timeLog.push_back(DATAMETRIC::GetLastInColumn("Time"));
                /* Save current throttle value to the throttle vector */
                    throttleLog.push_back(DATAMETRIC::GetLastInColumn("Throttle"));
                /* Save current brake value to the brake vector */
                    brakeLog.push_back(DATAMETRIC::GetLastInColumn("Brake"));
                    /* Save current steering angle value to the steer vector */
                 steer.push_back(DATAMETRIC::GetLastInColumn("Steering"));
```

```
}


  /* Add instructions to speed up to 60 MPH for the driver */
if( (sector != last_sector) && (sector == 5))                                   //line151
{
        if ((int(lap_number) + 1) % 2 == 0)
                /* Display image message on the screen to remind driver to speed up */
                DATAMETRIC::SetFeedbackImageEvent("speedup",3,0.5,0.7,1.0,0.2,0.8,0.7);
                /* Get initial time when the message appears */
                      initial_time = DATAMETRIC::GetLastInColumn("Time");
}
/* Display image when state changes */
if( (state != last_state) && (sector == 6 || sector == 5) )
{                                                                   //line161
        //cout << "state change" << endl;
        /* The vehicle state is brake_now */
        /* The vehicle state is brake_now */
            if (state == brake_now)
        {
                /* Get current time */
                      braking_stimulus_time = DATAMETRIC::GetLastInColumn("Time");

                if(kflag2){
                /* Display image on the screen */
DATAMETRIC::SetFeedbackImageEvent("stop_sign_panic",1,0.5,3.0,1.0,0.2,0.1,0.4);
//line171
                        kflag2 = false;
                }

        }
        /* Check if the vehicle state is braking */
        else if (state == braking)
        {
                /* Get current time */
                      braking_start_time = DATAMETRIC::GetLastInColumn("Time");
                /* Get current velocity */                                  //line181
                      braking_start_velocity = DATAMETRIC::GetLastInColumn("Velocity");
                //cout << "started braking at " << braking_start_time << endl;
                /* Get current vehicle position */
                pre_x = DATAMETRIC::GetLastInColumn("CarPositionX");
                pre_y = DATAMETRIC::GetLastInColumn("CarPositionY");
                pre_z = DATAMETRIC::GetLastInColumn("CarPositionZ");
        }
        /* The vehicle state is stopped */
            else if (state == stopped)
        {                                                           //line191
                kflag_velocity = true;
                /* Get current time */
                double current_time = DATAMETRIC::GetLastInColumn("Time");
                /* Get current velocity */
                      double current_velocity = DATAMETRIC::GetLastInColumn("Velocity");
                /* Calculate stopping_time */
```

```
                                        stopping_time = current_time - braking_start_time;
                                //reaction_time = braking_stimulus_time - braking_start_time;
                                /* Calcualte reaction time */
                                        reaction_time = braking_start_time - braking_stimulus_time;
                                //line201
                                reaction_time = int((reaction_time + 0.05) * 10)/10.0;
                                /* Calculate the decelartion */
                                        acceleration = ((current_velocity - braking_start_velocity) / stopping_time) /
                           9.81;
                                //double distance = braking_start_velocity * stopping_time + 0.5 * (-1) * acceleration
* stopping_time * stopping_time;
                                /* Get two digit number  */
                                stopping_time = int((stopping_time + 0.05) * 10)/10.0;
                                acceleration *= 2.2369; // convert to the real speed
                                acceleration = int((acceleration + 0.05) * 10)/10.0;                    //line211
                                //distance *= 2.2369; // convert to the real speed
                                //distance = int((distance + 0.05) * 10)/10.0;

// get the current location of the car
                                /* Get vehicle position */
                                post_x = DATAMETRIC::GetLastInColumn("CarPositionX");
                                post_y = DATAMETRIC::GetLastInColumn("CarPositionY");
                                post_z = DATAMETRIC::GetLastInColumn("CarPositionZ");
                                /* Get the distance between the pre and post vehicle position */
                                distance = sqrt( (post_x - pre_x) * (post_x - pre_x) + (post_y - pre_y) * (post_y -
pre_y) + (post_z - pre_z) * (post_z - pre_z) );                                    //line221
                                /* Convert distance unit from mph to km/h */
                                distance *= 2.2369; // convert to the real speed
                                /* Get last two digit number */
                                distance = int((distance + 0.05) * 10)/10.0;
                                //cout << "stopped braking at " << current_time << ", stopping time: " <<
                                stopping_time << ", delay: " << reaction_time << endl;
                                /* Create a string */
                                ostringstream os;
                                //cout.setw(2);
                                /* Store information in this string */                              //line231
                                os << "You stopped in " << stopping_time << " seconds and " << distance <<" feet "
<< endl;
                                os << "after a " << reaction_time << " second delay" << endl;
                                os << "with a deceleration of " << -acceleration << " Gs. " << endl;
                                os << "  - Stopped in " << stopping_time << " seconds\n";
                                os << "  - " << distance << " feet stopping distance\n";
                                os << "  - " << -acceleration << " Gs deceleration rate"<< endl;
                                /* Display message on the screen */
                                DATAMETRIC::SetFeedbackMessageEvent("Full Stop Braking Report", os.str(),
3.0, 1.0, 2.0);                                                           //line241
                                //DATAMETRIC::SetFeedbackImageEvent("info", 9,0.2,2.0,1.0,0.2,0.8,0.1);
                                /* Add score list here */
                                score3_1 = 10.0;
                                /* Assign score3_2 depends on the reaction time */
                                if(reaction_time < 1)
                                  score3_2 = 5.0;
                                if(distance < 70)
```

```
                  score3_3 = 5.0;
                if(stopping_time < 5)
                  score3_4 = 5.0;                                         //line251
                  /* Sum the total score by each item score */
                //score3 = score3_1 + score3_2 + score3_3 + score3_4;
                    /* We collect data till the vehicle has completely stopped */
                /* Check if the vehicle has stopped */
                    kkflag = true;
                /* Declear a time stream */
                    stringstream timeStream;
                    /* Iterator time from beginng to the end */
                for(std::vector<double>::iterator itTime = timeLog.begin(); itTime != timeLog.end();
                                                                    //line261
                      ++itTime) {
                  timeStream << "," << (*itTime - initial_time);
                }
                /* Open file and save stream */
                ofstream outFileTime("/home/vdrift/Desktop/3rd/timeFor3rdScenario.dat");
                if(outFileTime.is_open()) {
                      /* Send information to these file */
                          outFileTime << timeStream.str();
                }
                /* Close file */                                          //line271
                outFileTime.close();

                /* Declear a string */
                stringstream velocityStream;
                /* Iterate velocity from beginning to the end */
                for(std::vector<double>::iterator itVelocity = velocityLog.begin(); itVelocity !=
velocityLog.end(); ++itVelocity) {
                  /* Get minimal velocity */
                  if(min_velocity > *itVelocity) min_velocity = *itVelocity;
                  velocityStream << "," << *itVelocity;
                }                                                     //line281

/* Open file */
ofstream outFileVelocity("/home/vdrift/Desktop/3rd/velocityFor3rdScenario.dat");
                if(outFileVelocity.is_open()) {
                  outFileVelocity << velocityStream.str();
                }
                /* Close file */
                outFileVelocity.close();
                    /* Set throttle stream*/
                stringstream throttleStream;
                /* Iterate throttle from beginning to the end */             //line291
                for(std::vector<double>::iterator itThrottle = throttleLog.begin(); itThrottle !=
throttleLog.end(); ++itThrottle) {
                 /* Save throttle value to the throttle stream */
                  throttleStream << "," << *itThrottle;
                }
/* Open file */
                ofstream outFileThrottle("/home/vdrift/Desktop/3rd/throttleFor3rdScenario.dat");
    if(outFileThrottle.is_open()) {
```

```
                          outFileThrottle << throttleStream.str();                    //line301
                        }
                        /* Close file */
                        outFileThrottle.close();

                            /* Brake Part */
                        stringstream brakeStream;
                        for(std::vector<double>::iterator itBrake = brakeLog.begin(); itBrake !=
brakeLog.end(); ++itBrake) {
                          brakeStream << "," << *itBrake;
                        }
                        /* Open file and save data */                          //line311
                        ofstream outFileBrake("/home/vdrift/Desktop/3rd/brakeFor3rdScenario.dat");
                        if(outFileBrake.is_open()) {
                          outFileBrake << brakeStream.str();
                        }
                        /* Close data */
                        outFileBrake.close();
                            /*Add score */
                        ifstream inFile;
                        inFile.open("dataTest.csv");
                        /* Open file and save data */                          //line321
                        if(!inFile) {
                          std::cout << "Open initialDataTest.csv" << std::endl;
                          inFile.open("initialDataTest.csv");
                        }
                        /* Read file line by line */
                        while(!inFile.eof()) {
                          string oneLine;
                         /* Get each line */
                          getline(inFile,oneLine);
                        /* Put each line to the back of vector */                    //line331
                            v.push_back(oneLine);
                        }
                        /* Close file */
                        inFile.close();
                        /* Reallocate the size of vector */
                        v.resize(100);
                            /* Weight of checking vehicle if it has stopped */
                        int weight1 = 7;
                        /* Weight of reaction time */
                        int weight2 = 3;                                             //line341
                        /* Weight of stopping distance */
                            int weight3 = 3;
                            /* Weight of stopping time */
                        int weight4 = 3;
                        /* Weight of deceleration */
                        int weight5 = 3;
                        /* Weight of steering angle */
                        int weight6 = 6;
                        /* Weight of lateral position */
                        //int weight7 = 3;                                           //line351
```

```
        /*Check if the car has stopped */
if(kflag_velocity) score_checkStopped = 4;
else score_checkStopped = 0;
/* Multiple weight with its related score */
score_checkStopped *= weight1;
/* Declear stream s_check */
stringstream s_check;
s_check << score_checkStopped << "/28;";
/* Add stream to the vector */
v[15] += s_check.str();                                       //line361
        /* Reaction Time */
if(reaction_time <= 1) score_reaction_time = 4;
else if(reaction_time > 1 && reaction_time <= 1.5) score_reaction_time = 3;
else if(reaction_time > 1.5 && reaction_time <= 2) score_reaction_time = 2;
else if(reaction_time > 2 && reaction_time <= 2.5) score_reaction_time = 1;
else score_reaction_time = 0;
score_reaction_time *= weight2;
stringstream s_reaction_time;
s_reaction_time << score_reaction_time << "/12;";
v[16] += s_reaction_time.str();                               //line371
        /* Stopping Distance */
        /* If the stopping distance is less than 70 km, then get full credits */
if(distance <= 70) score_stopping_distance = 4;
/* If the stopping distance is between 70 and 100 km, get partial credits */
else if(distance > 70 && distance <= 100) score_stopping_distance = 3;
/* If the stopping distance is between 100 and 150 km, get partial credits */
        else if(distance > 100 && distance <= 150) score_stopping_distance = 2;
        /* If the stopping distance is between 150 and 200 km, get partial credits */
else if(distance > 150 && distance < 200) score_stopping_distance = 1;
/* If the distance is over 200km, no credits will be given */            //line381
        else score_stopping_distance = 0;
        /* Multiply scoring distance with weight3 */
score_stopping_distance *= weight3;
/* Allocate a string space to contain distance information */
stringstream s_distance;
s_distance << score_stopping_distance << "/12;";
/* Save distance parameter to the vector */
v[17] += s_distance.str();
        /* Stopping Time */
        /* if the stopping time is less than 2 second, then get full credits */ //line391
if(stopping_time <= 2) score_stopping_time = 4;
/* if the stopping time is between 2 and 2.5, get partial credits */
else if(stopping_time > 2 && stopping_time <= 2.5) score_stopping_time = 3;
/* if the stopping time is between 2.5 and 3, get partial credits */
        else if(stopping_time > 2.5 && stopping_time <= 3) score_stopping_time = 2;
/* if the stopping time is between 3 and 4, get partial credits */
        else if(stopping_time > 3 && stopping_time < 4) score_stopping_time = 1;
/* if the stopping time is over 4 second, no credits is given */
        else score_stopping_time = 0;
        /* Multiply score of stopping time with weight 4 */               //line401
score_stopping_time *= weight4;
/* Allocate string stopping_time */
stringstream s_stopping_time;
```

```
s_stopping_time << score_stopping_time << "/12;";
/* Save stopping time string to vector */
v[18] += s_stopping_time.str();
        /* Acceleration */
/* Get the absolution of the acceleration */
        double delta = abs( abs(acceleration) - 0.7);
        /* If the acceleration is less than 0.1 gs, get full credits */              //line411
if(delta <= 0.1) score_acceleration = 4;
/* If the acceleration is between 0.1 and 0.3 gs, get partial credits */
else if(delta > 0.1 && delta <= 0.3) score_acceleration = 3;
/* If the acceleration is between 0.1 and 0.3 gs, get partial credits */
        else if(delta > 0.3 && delta <= 0.5) score_acceleration = 2;
/* If the acceleration is between 0.1 and 0.3 gs, get partial credits */
        else if(delta > 0.5 && delta <= 1.0) score_acceleration = 1;
/* If the acceleration is over 1.0 gs, no credits is given */
        else score_acceleration = 0;
        /* Multiply the score of acceleration with weight 5 */
//line421
score_acceleration *= weight5;
/* Allocate the string s */
stringstream s_acceleration;
s_acceleration << score_acceleration << "/12;";
/* Allocate acceleration string to vector */
v[19] += s_acceleration.str();
        /* Steer Sum and Max */

        /* Iterate steering angle vector from beginning to the end */
for(std::vector<double>::iterator itSteer = steer.begin(); itSteer != steer.end();
++itSteer) {                                                                      //line431
/* Sum all the steering angle */
        sumSteer += *itSteer;
  if(maxSteer < *itSteer) maxSteer = *itSteer;
}
/* Get absolution of steering angle */
        sumSteer = abs(sumSteer);
        /* If the sum of steering angle is less than 1500 degree and the maximum
           Steering angle is less than 90, then get full credits */
if(sumSteer <= 1500 && maxSteer <= 90) score_steer = 4;
        /* If the sum of steering angle is less than 1500 degree but the maximum
           Steering angle is less than 90, then get parital credits */              //line441
else if(sumSteer <= 1500 && maxSteer > 90) score_steer = 3;
        /* If the sum of steering angle is over than 1500 degree but the maximum
           Steering angle is less than 90, then get parital credits */
        else if(sumSteer > 1500 && maxSteer <= 90) score_steer = 2;
        /* If none of the above, get one credit */
else score_steer = 1;
/* Multiply the steering score with weight6 */
score_steer *= weight6;
/* Apply steering angle string */
stringstream s_steer;                                          //line451
s_steer << score_steer << "/24;";
/* Assign string steer to vector */
v[20] += s_steer.str();
```

```
                                /* Save data and send to the dataTest.csv */
                            std::vector<string>::iterator it;
                            /* Open file */
                            ofstream outFile("dataTest.csv");
                            if(outFile.is_open()) {
                              for(it = v.begin(); it != v.end(); ++it) {
                                outFile << *it << std::endl;                          //line461
                              }
                            }
                            /* Close file */
                            outFile.close();
                                                /* Display face here */
            /*   double tempScore = score_checkStopped + score_reaction_time +
              score_stopping_distance + score_stopping_time +
              score_acceleration;
               std::cout << "the temp score is: " << tempScore << std::endl;
            /* If the score is over 90, then display a simle face on the screen */             //line471
               if(tempScore >= 90)
                        DATAMETRIC::SetFeedbackImageEvent("smile_sign",6,0.2,2.0,1.0,0.2,0.8,0.7);
              /* If the score is between 70 and 90, display a ok image on the screen */
               else if(tempScore >= 70 && tempScore < 90)
                        DATAMETRIC::SetFeedbackImageEvent("ok_sign",7,0.2,2.0,1.0,0.2,0.8,0.7);
               Else
                       /* If the score is lower than 70, display a cry face on the screen */
                        DATAMETRIC::SetFeedbackImageEvent("cry_sign",8,0.2,2.0,1.0,0.2,0.8,0.7);

                       }
                       /* If the vehcle state is not braking , start start time as negative */        //line481
                       else if (state == not_braking)
                       {
                            //cout << "not braking" << endl;
                            braking_start_time = -1.0;
                       }
            }
  //printf(%.1f,&)
            /* Record the current vehicle to last state */
            last_state = state;
            /* Sum the score of each item */                                          //line491
            score3 = score_checkStopped + score_reaction_time +
            score_stopping_distance + score_stopping_time +
            score_acceleration;
            /* Normalized the final score */
               score3 /= 4;
               /* Sned all the data to the database, here is an excel file */
            output_data["score3_1"] = score_checkStopped;
            output_data["score3_2"] = score_reaction_time;
            output_data["score3_3"] = score_stopping_distance;
            output_data["score3_4"] = score_stopping_time;                            //line501
            output_data["score3_5"] = score_acceleration;
            output_data["score3"] = score3;
            output_data["param3_1"] = stopping_time;
            output_data["param3_2"] = distance;
```

```
output_data["param3_3"] = reaction_time;
output_data["param3_4"] = acceleration;
}
```

%Section 7-   Header file of “Animal Avoidance Module”

```
#ifndef ANIMALAVOIDMETRIC_H                                                          //line1
#define ANIMALAVOIDMETRIC_H
#include <vector>
#include "datamanager.h"

/* This is the place where you should describe your new DATAMETRIC- derived class. */
class ANIMALAVOIDMETRIC : public DATAMETRIC
{
        public:
                /* Constructor */
                ANIMALAVOIDMETRIC(DATAMETRIC_CTOR_PARAMS_DEF);
                /* Destructor */                                                 //line11
                ~ANIMALAVOIDMETRIC();
                /* Update the calculations, add to log, generate events */
                void Update(float dt);
        protected:
        private:
                static METRICTYPEREGISTER<ANIMALAVOIDMETRIC> reg; //!< object registering
this type for creation by name
                /* Declear a parameter to detect lane position */
                    double laneScore;
                /* Declear a parameter to detect speed */                        //line21
                    double speedScore;
                /* Declear a parameter to detect the distance between the vehicle and the center line */
                    double distanceScore;
                /* Declear a parameter to calculate yaw rate and its score */
                    double yawRateScore;
                    /* Declear a parameter to give a score */
                double score4;
                /* Declear a parameter to get the sum of steering angle */
                double sumAngle;
                /* Declear a parameter to get the peak of steering angle */              //line31
                double peakSteeringAngle;
                /* This parameter is not used */
                    double countVelocity;
                    /* Declear a parameter to get the average of velocity during this scenario */
                double averageVelocity;
                /* Declear a parameter to sum all the velocity during this scenario */
                    double sumVelocity;
                    /* Declear a parameter to get the peak velocity */
                double peakVelocity;
                /* Declear a parameter to get current brake value */                     //line41
                double brake;
                /* Declear a parameter to get current throttle value */
                    double throttle;
                /* Declear a parameter to get lateral position */
                double lateralPosition;
                /* Declear a parameter to get current velocity */
                double velocity;
                /* Create a v vector */
                    std::vector<string> v;
                    /* Createa a score for vector */                         //line51
                int score_velocity;
                /* Declear a parameter to check if the vehicle is on the road */
                bool onRoad;
                /* Declear a parameter to check if the vehicle is driving on the right lane */
                bool rightLane;
                /* Get a score based on different position */
                int score_position;
                /* Get a score based on different steering angle */
                int score_steer;
                /* Create a time vector */
                std::vector<double> timeLog;
                /* Create an initial time value */
                double initialTime;
                /* Create a velocity vector */
                std::vector<double> velocityLog;
                /* Create a throttle vector */
                std::vector<double> throttleLog;
                /* Create a brake vector */
                std::vector<double> brakeLog;
                /* Create a steering angle vector */
                std::vector<double> steerLog;

};

#endif // ANIMALAVOIDMETRIC_H
```

%Section 8-   Implement file of “Animal Avoidance Module”

```
#include <iostream>
#include <assert.h>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <fstream>
using std::cout;
using std::endl;
using std::stringstream;
using std::ofstream;
using std::ifstream;
using std::fstream;
using std::string;

#include "datametric_animal_avoid.h"

/// static class member declaration, register metric type name
METRICTYPEREGISTER<ANIMALAVOIDMETRIC>
ANIMALAVOIDMETRIC::reg("AnimalAvoid");
/* Constructor */
ANIMALAVOIDMETRIC::ANIMALAVOIDMETRIC(DATAMETRIC_CTOR_PARAMS_DEF)
 : DATAMETRIC(DATAMETRIC_CTOR_PARAMS),
  /* Initial all the parameters with value zero */
  laneScore(0.0),
  speedScore(0.0),
  distanceScore(0.0),
  yawRateScore(0.0),
  score4(0.0),
  sumAngle(0.0),
  peakSteeringAngle(0.0),
  countVelocity(0.0),
  averageVelocity(0.0),
  sumVelocity(0.0),
  peakVelocity(0.0),
  brake(0.0),
  throttle(0.0),
  lateralPosition(0.0),
  velocity(0.0),
  v(),
  score_velocity(0),
  onRoad(false),
  rightLane(false),
  score_position(0),
  score_steer(0),
  timeLog(),
  initialTime(0.0),
  velocityLog(),
  throttleLog(),
  brakeLog(),
  steerLog()
{
        // Enabled by default
        DATAMETRIC::run = true;                                          //line51
}
/* Destructor */
ANIMALAVOIDMETRIC::~ANIMALAVOIDMETRIC()
{
        // Clean up any class data
}
/* Call update function */
void ANIMALAVOIDMETRIC::Update(float dt)
{
        // Do nothing when disabled                                      //line61
        if (!run)
                return;
        /* Get current sector in the lap */
        DATALOG::log_data_T sector = DATAMETRIC::GetLastInColumn("Sector");
        /* Get last sector in the lap */
          DATALOG::log_data_T last_sector = DATAMETRIC::GetNextLastInColumn("Sector");
        /* Get value indicating whether the vehicle is currently on the left or right of the lane */
          DATALOG::log_data_T value = DATAMETRIC::GetLastInColumn("LeftOrRight");
        /* Get lap number */                                             //line71
          DATALOG::log_data_T lap_number = DATAMETRIC::GetLastInColumn("LapNumber");
        /* Get last velocity */
          velocity = DATAMETRIC::GetLastInColumn("Velocity");
        /* Get the distance between car and centerline */
          DATALOG::log_data_T distance =
DATAMETRIC::GetLastInColumn("DistanceBetweenCarAndTrack");
          /* Get current yaw rate */
        DATALOG::log_data_T yaw_rate = DATAMETRIC::GetLastInColumn("YawRate");
        /* Declear two strings */
        stringstream ss;                                                         //line81
        stringstream yy;
        /* Select all the parameters in this scenario */
        if( (sector != last_sector) && (sector == 7) )
        {
                if ((int(lap_number) + 1) % 2 == 0)
                        DATAMETRIC::SetFeedbackImageEvent("animal",5,0.5,2.0,1.0,0.2,0.7,0.1);
                /* Get the array of all time value in this scenario */
                    initialTime = DATAMETRIC::GetLastInColumn("Time");
        }
        /* Collect data during the whole sector 7 */                             //line91
        if(sector == 7) {
          /* Get Sum of the Steering Angle */
          sumAngle += abs(DATAMETRIC::GetLastInColumn("Steering"));
          /* Get Peak Value of the Steering Angle */
          if( peakSteeringAngle < DATAMETRIC::GetLastInColumn("Steering")) {
            peakSteeringAngle = DATAMETRIC::GetLastInColumn("Steering");
          }
          peakSteeringAngle = int((peakSteeringAngle + 0.05) * 10) / 10.0;
          /* Get Sum of Velocity */
          sumVelocity += DATAMETRIC::GetLastInColumn("Velocity");                        //line101
          countVelocity++;
          /* Get Peak of Velocity */
```

```
   if( peakVelocity < DATAMETRIC::GetLastInColumn("Velocity")) {
      peakVelocity = DATAMETRIC::GetLastInColumn("Velocity");
   }
  /* Get last two digits of a number */
   peakVelocity = int((peakVelocity + 0.05) * 10) / 10.0;
   /* Collect Data for Matlab */
  /* Get all the time and save them ine array */
   timeLog.push_back(DATAMETRIC::GetLastInColumn("Time"));                    //line111
   /* Get all the velocity and save them in the array */
     velocityLog.push_back(DATAMETRIC::GetLastInColumn("Velocity"));
   /* Get all the throttle value and save them in the array */
     throttleLog.push_back(DATAMETRIC::GetLastInColumn("Throttle"));
   /* Get all the brake value and save them in the array */
     brakeLog.push_back(DATAMETRIC::GetLastInColumn("Brake"));
   /* Get all the steering angle value and save them in the array */
   steerLog.push_back(DATAMETRIC::GetLastInColumn("Steering"));
}

if( (sector != last_sector) && (sector == 8) )                                 //line121
{
    /* Get Average of Velocity */
    averageVelocity = sumVelocity / countVelocity;
    /* Get last two digits of number */
       averageVelocity = int((averageVelocity + 0.05) * 10) / 10.0;

    /* Get Throttle and Brake */
    brake = DATAMETRIC::GetLastInColumn("Brake");
    /* Get last two digits of number */
       brake = int((brake + 0.05) * 10) / 10.0;
    throttle = DATAMETRIC::GetLastInColumn("Throttle");                        //line131
    throttle = int((throttle + 0.05) * 10) / 10.0;

           /* Output Value */
           cout << "************ Scenario 4 **********************" << endl;
           cout << "The sum of steering angle is: " << sumAngle << endl;
           cout << "The peak of steering angle is: " << peakSteeringAngle << endl;
           cout << "The average velocity is: " << averageVelocity << endl;
           cout << "The peak velocity is: " << peakVelocity << endl;
           cout << "The brake is: " << brake << endl;
           cout << "The throttle is: " << throttle << endl;
           cout << "Distance between car and track is: " << distance << endl;  //line141
           cout << "************** End *************************" << endl;



            /* Test the lab_number is even or odd */
                if ((int(lap_number) + 1) % 2 == 0)
           {
              /* Get the distance between car and centerline of the track */
              lateralPosition = DATAMETRIC::GetLastInColumn("DistanceBetweenCarAndTrack");
                          /* Get last two digits of the number */
                   lateralPosition = int((lateralPosition + 0.05) * 10) / 10.0;
              /* Test if the vehicle is on the right lane */                   //line151
```

```
if(value == -1)
       {
             ss << "Correct Lane" << endl;
            /* Get points */
                   laneScore= 5.0;
       }
       else
            ss << "Wrong Lane" << endl;
       /* If the velocity is less than 30 mph */
           if(velocity < 30)                                                    //line161
       {
            /* Get points */
                   speedScore = 10.0;
            ss << "Speed OK" << endl;
       }
       else
            /* Display message: speeding */
                   ss << "Speeding!" << endl;
       /* Detect the distance between vehicle and centerline */
       if(distance >= 5 && distance < 10)                                 //line171
       {
            /* Get points */
                   distanceScore = 5.0;
            ss << "Drive on the correct lane" << endl;
       }
       else
            /* Display warning message */
                   ss << "Warning! not on the road" << endl;
       /* Detect the value of yaw rate */
           if(abs(yaw_rate) < 50)                                         //line181
       {
            /* Get yaw rate score */
                   yawRateScore = 5.0;
            ss << "Drive smoothly" << endl;
       }
       else
       {
            ss << "Drive in a danger way" << endl;
       }                                                            //line191

       //DATAMETRIC::SetFeedbackMessageEvent("animal_avoid",ss.str(),2.0,1.0,0.2);


           /* Create and Load dataTest.csv */
       ifstream inFile;
       /* Open dataTest.csv file */
       inFile.open("dataTest.csv");
       /* Check if the file has been opened successfully */
       if(!inFile) {
         std::cout << "Open initialDataTest.csv" << std::endl;
         inFile.open("initialDataTest.csv");
       }                                                            //line201
       /* Get information line by line from the file */
```

```
                while(!inFile.eof()) {
                  string oneLine;
                  getline(inFile, oneLine);
                /* Save each line to a container, here it is vector v */
                  v.push_back(oneLine);
                }
                /* Close file */
                inFile.close();
                /* Change size of the velocity */                              //line211
                v.resize(100);
                        /* Get weight for each scenario */
                int weight1 = 5;
                int weight2 = 15;
                int weight3 = 5;

                        /* Give score of the velocity based on the various speed */
                if(velocity <= 20) score_velocity = 4;
                else if(velocity > 20 && velocity <= 25) score_velocity = 3;
                else if(velocity > 25 && velocity <= 40) score_velocity = 2;
      else if(velocity > 40 && velocity <= 50) score_velocity = 1;                        //line221
                else score_velocity = 0;
                score_velocity *= weight1;
                /* Declare a string to store the score of the vehicle velocity */
                        stringstream s_velocity;
                s_velocity << score_velocity << "/20;";
                /* Add score to v container */
                v[22] += s_velocity.str();

                        /* Get the number of wheels that are on the road */
           if(DATAMETRIC::GetLastInColumn("offRoad") == 4) onRoad = true;      //line231
                /* Give score based on two factors, one is the number of wheels on the road
                   The other is the position of the vehicle, detect whether it is on the right lane */
                        if(DATAMETRIC::GetLastInColumn("LeftOrRight") == -1) rightLane = true;
                if(rightLane && onRoad) score_position = 4;
                else if(rightLane && !onRoad) score_position = 3;
                else if(!rightLane && onRoad) score_position = 2;
                else score_position = 1;
                /* Multiply the position score with its related weight */
                score_position *= weight2;
                /* Declear a string to store position parameter */             //line241
                stringstream s_position;
                s_position << score_position << "/60;";
                /* Add position element to the vector */
                v[23] += s_position.str();
                        /* Get score of steering angle to detect if the student driver the vehicle
                        smoothly*/
                if(sumAngle <= 1800 && peakSteeringAngle <= 90) score_steer = 4;
                else if(sumAngle <= 1800 && peakSteeringAngle > 90) score_steer = 3;
                else if(sumAngle > 1800 && peakSteeringAngle <= 90) score_steer = 2;
                else score_steer = 1;                                       //line251
                /* Multiplay the score of the steering angle with its related weight */
                score_steer *= weight3;
                /* Declear a string to store the steering angle parameter */
```

```
                    stringstream s_steer;
                    s_steer << score_steer << "/20;";
                    /* Add element to the vector */
                    v[24] += s_steer.str();
                        /* Send v to the dataTest.csv */
                    std::vector<string>::iterator it;
                    ofstream outFile("dataTest.csv");                              //line261
                    if(outFile.is_open()) {
                      for(it = v.begin(); it != v.end(); ++it) {
                        outFile << *it << std::endl;
                      }
                    }
  /* Get the final score by summing the four scores */
  double tempScore = score_velocity + score_position + score_steer;
  //std::cout << "the temp score is: " << tempScore << std::endl;
/* Display face images based on the score the student get */
  if(tempScore >= 90)                                                              //line271
        DATAMETRIC::SetFeedbackImageEvent("smile_sign",6,0.2,2.0,1.0,0.2,0.6,0.1); // 0.8 0.7
  else if(tempScore >= 70 && tempScore < 90)
        DATAMETRIC::SetFeedbackImageEvent("ok_sign",7,0.2,2.0,1.0,0.2,0.6,0.1);
  else
        DATAMETRIC::SetFeedbackImageEvent("cry_sign",8,0.2,2.0,1.0,0.2,0.6,0.1);

                /* Send data to Matlab */

                        /* Read all the time data from the database*/
                    stringstream timeStream;
                    for(std::vector<double>::iterator itTime = timeLog.begin();        //line281
                            itTime != timeLog.end(); ++itTime) {
                      timeStream << "," << (*itTime - initialTime);
                    }
                    /* Write all the time data to a matlab type file, and point the location */
                    ofstream outFileTime("/home/vdrift/Desktop/4th/timeFor4thScenario.dat");
                    /* Open the matlab type file and write the time data */
                        if(outFileTime.is_open()) {
                      outFileTime << timeStream.str();
                    }
                    /* Close the file data */                                      //line291
                    outFileTime.close();
                        /* Save all velocity data and write to a matlab dat file */
                    stringstream velocityStream;
                    for(std::vector<double>::iterator itVelocity = velocityLog.begin();
                            itVelocity != velocityLog.end(); ++itVelocity) {
                      velocityStream << "," << *itVelocity;
                    }
                    /* Open and save file to a pointed location */
                    ofstream outFileVelocity("/home/vdrift/Desktop/4th/velocityFor4thScenario.dat");
                    //line301
                    /* Write file */
                        if(outFileVelocity.is_open()) {
                      outFileVelocity << velocityStream.str();
                    }
                    /* Close file */
                outFileVelocity.close();
                    /* Read all the throttle data from database */
                stringstream throttleStream;
                for(std::vector<double>::iterator itThrottle = throttleLog.begin();
                      itThrottle != throttleLog.end(); ++itThrottle) {              //line311
                  throttleStream << "," << *itThrottle;
                }
                /* Write all the data to a pointed matlab type file */
                ofstream outFileThrottle("/home/vdrift/Desktop/4th/throttleFor4thScenario.dat");
                if(outFileThrottle.is_open()) {
                  outFileThrottle << throttleStream.str();
                }
                /* Close the file */
                outFileThrottle.close();                                            //line321
                    /* Read all the brake data from the database */
                stringstream brakeStream;
                for(std::vector<double>::iterator itBrake = brakeLog.begin();
                      itBrake != brakeLog.end(); ++itBrake) {
                  brakeStream << "," << *itBrake;
                }
                /* Write the brake data to a matlab dat type file */
                ofstream outFileBrake("/home/vdrift/yaoqimin/matlab/bin/brakeFor4thScenario.dat");
                if(outFileBrake.is_open()) {                                   //line331
                  outFileBrake << brakeStream.str();
                }
                /* Close the file */
                outFileBrake.close();
                    /* Read all the steering angle data from the database */
                stringstream steerStream;
                for(std::vector<double>::iterator itSteer = steerLog.begin();
                      itSteer != steerLog.end(); ++itSteer) {
                  steerStream << "," << *itSteer;
                }                                                              //line341
                /* Save and write the steering angle data to a matlab dat type file */
                ofstream outFileSteer("/home/vdrift/Desktop/4th/steerFor4thScenario.dat");
                if(outFileSteer.is_open()) {
                  outFileSteer << steerStream.str();
                }
                /* Close file */
                outFileSteer.close();
        }
        else
        {                                                                      //line351
                /* Display test messages on the screen */
                    yy << "Slow down under 30 mph" << endl;
                yy << "and drive on the right lane" << endl;
//              DATAMETRIC::SetFeedbackMessageEvent("AAmessage",yy.str(),2.0,1.0,0.2);
        }
}

/* Write the score data to the database */
double score4_3 = distanceScore;
/* Get the total score */
```

```
            score4 = score_velocity + score_position + score_steer;                               //line361
          /* Normalize the total score */
            score4 /= 4;
            /* Save the score of each item to database */
          output_data["score4_1"] = score_velocity;
          output_data["score4_2"] = score_position;
          output_data["score4_3"] = score4_3;
          output_data["score4_4"] = score_steer;
          output_data["score4"] = score4;
/* Save the vehicle dynamic parameters to the database */
  output_data["param4_1"] = peakVelocity;                                                           //line371
  output_data["param4_2"] = averageVelocity;
  output_data["param4_3"] = sumAngle;
  output_data["param4_4"] = peakSteeringAngle;
  output_data["param4_5"] = throttle;
  output_data["param4_6"] = brake;
  output_data["param4_7"] = lateralPosition;}
```

%Section 9-   Header file of Scoring System

```
#ifndef TOTALSCOREMETRIC_H                                              //line1
#define TOTALSCOREMETRIC_H

#include "datamanager.h"

/*This class aims to statistic the total score and rete students' driving performances. */
class TOTALSCOREMETRIC : public DATAMETRIC
{
        public:
                /* Constructor */
                TOTALSCOREMETRIC(DATAMETRIC_CTOR_PARAMS_DEF);
                /* Destructor */
                ~TOTALSCOREMETRIC();                                    //line11
                /* Update the calculations, add to log, generate events */
                void Update(float dt);
        protected:
        private:
                    /* object registering this type for creation by name */
                    static METRICTYPEREGISTER<TOTALSCOREMETRIC> reg;
                /* Declear a total score */
                    double totalscore1;
                    /* Declear a current vehicle velocity */
                double detectCurrentVelocity;                           //line21
                /* Declear a vehicle velocity one time earlier than the current vehicle velocity */
                    double detectLastVelocity;
                    /* Dclear the number of times the vehicle is driving over the speed limit */
                int numOfSpeeding;
                int numCompare1;
                int numCompare2;
                /* Create a vector container to store all the vehicle velocities in the lap */
                //vector<double> detectVelocityLog;
                /* Declear a parameter to store the number of times the vehicle wheels are on the road */
                double currentDrivingOffRoad;                           //line31
                double lastDrivingOffRoad;
                int numDrivingOffRoad;
                /* Declear a parameter to store the number of times the vehicle is driving across
                   the double yellow line */
                double currentDrivingAcrossLine;
                double lastDrivingAcrossLine;
                int numDrivingAcrossLine;
};

#endif // TOTALSCOREMETRIC_H
```

%Section 10-   Implement file of Scoring System

```
#include <iostream>                                                //line1
#include <assert.h>
#include <sstream>
#include <fstream>
#include <vector>
#include <cstdlib>
#include <string>
#include "carwheelposition.h"
#include "bezier.h"
using std::cout;
using std::endl;                                                         //line11
using std::stringstream;
using std::ofstream;
using std::fstream;
using std::ifstream;
using std::string;
using std::vector;

#include "datametric_total_score.h"

/* static class member declaration, register metric type name */
METRICTYPEREGISTER<TOTALSCOREMETRIC> TOTALSCOREMETRIC::reg("TotalScore");

/* Initialization list */                                          //line21
TOTALSCOREMETRIC::TOTALSCOREMETRIC(DATAMETRIC_CTOR_PARAMS_DEF)
 : DATAMETRIC(DATAMETRIC_CTOR_PARAMS),
  totalscore1(0),
  detectCurrentVelocity(0.0),
  detectLastVelocity(0.0),
  numOfSpeeding(0),
  numCompare1(0),
  numCompare2(0),
  currentDrivingOffRoad(0.0),
  lastDrivingOffRoad(0.0),                                              //line31
  numDrivingOffRoad(0),
  currentDrivingAcrossLine(0.0),
  lastDrivingAcrossLine(0.0),
  numDrivingAcrossLine(0)
  //detectVelocityLog()
{
	// Enabled by default
	DATAMETRIC::run = true;
}
/* Destructor function in the class */                                 //line41
TOTALSCOREMETRIC::~TOTALSCOREMETRIC()
{
	// Clean up any class data
}

/* Update function in the class */
void TOTALSCOREMETRIC::Update(float dt)
{
        // Do nothing when disabled
        if (!run)
                return;                                                         //line51

        //DATALOG::log_data_T lap_number = DATAMETRIC::GetLastInColumn("LapNumber");
        // Get columns out
        /* Get four scores, each score represents a scenario */
        DATALOG::log_column_T const* score1_col = DATAMETRIC::GetColumn("score1");
        DATALOG::log_column_T const* score2_col = DATAMETRIC::GetColumn("score2");
        DATALOG::log_column_T const* score3_col = DATAMETRIC::GetColumn("score3");
        DATALOG::log_column_T const* score4_col = DATAMETRIC::GetColumn("score4");
        /* Get lap sector from the database */
          DATALOG::log_data_T sector = DATAMETRIC::GetLastInColumn("Sector");
        DATALOG::log_data_T last_sector = DATAMETRIC::GetNextLastInColumn("Sector");
        /* Get the last score from each log */                                          //line61
        DATALOG::log_data_T s1 = 0.0;
        DATALOG::log_data_T s2 = 0.0;
        DATALOG::log_data_T s3 = 0.0;
        DATALOG::log_data_T s4 = 0.0;
        s1 = score1_col->back();
        s2 = score2_col->back();
        s3 = score3_col->back();
        s4 = score4_col->back();
        /* When the vehicle is driving through the end point of the track. A final score will display on the
screen */                                                                               //line71
        if(sector != last_sector)
                cout << "The final score is: " << s1 << "+" << s2 << "+" << s3 << "+" << s4 << "=" <<
totalscore1 <<endl;
          /* Get velocity data from the database */
        detectCurrentVelocity = DATAMETRIC::GetLastInColumn("Velocity");
        detectLastVelocity = DATAMETRIC::GetNextLastInColumn("Velocity");
        /* Convert the velocity unit from mph to km/h */
          detectCurrentVelocity *= 2.2369;
        detectLastVelocity *= 2.2369;
          /* Check and give warnings if the vehicle is speeing on the 1st scenario */           //line81
        if(sector == 1 || sector == 2) {
        /* Detect if the velocity of the vehicle is just across 40 km/h */
          if(detectLastVelocity <= 40 && detectCurrentVelocity > 40) {
                numCompare1++;
                /*  Display warning message on the screen */
                DATAMETRIC::SetFeedbackMessageEvent("Speeding", "-5",0.5,0.5,0.5);
          }
          /* Detect if the velocity of the vehicle is just across 45 km/h */
          else if(detectLastVelocity <= 45 && detectCurrentVelocity > 45) {
                numCompare1++;                                                  //line91
                /* Display the warning message on the screen */
                DATAMETRIC::SetFeedbackMessageEvent("Speeding", "-10",0.5,0.5,0.5);
          }
        /* Detect if the velocity of the vehicle is just across 55 km/h */
          else if(detectLastVelocity <= 55 && detectCurrentVelocity > 55) {
                numCompare1++;
                /* Display the warning message on the screen */
                DATAMETRIC::SetFeedbackMessageEvent("Speeding", "-15",0.5,0.5,0.5);
```

```
  }
  else { }                                                              //line101

 }
 /* Check and give warnings if the vehicle is speeing on the 2nd scenario */
 else if(sector == 3 || sector == 4) {
 /* Detect if the velocity of the vehicle is just across 40 km/h */
   if(detectLastVelocity <= 40 && detectCurrentVelocity > 40) {
        numCompare1++;
        /* Display the warning message on the screen */
        DATAMETRIC::SetFeedbackMessageEvent("Speeding", "-5",0.5,0.5,0.5);
   }
 /* Detect if the velocity of the vehicle is just across 45 km/h */                    //line111
   else if(detectLastVelocity <= 45 && detectCurrentVelocity > 45) {
        numCompare1++;
        /* Display the warning message on the screen */
        DATAMETRIC::SetFeedbackMessageEvent("Speeding", "-10",0.5,0.5,0.5);
   }
 /* Detect if the velocity of the vehicle is just across 55 km/h */
   else if(detectLastVelocity <= 55 && detectCurrentVelocity > 55) {
        numCompare1++;
        /* Display the warning message on the screen */
        DATAMETRIC::SetFeedbackMessageEvent("Speeding", "-15",0.5,0.5,0.5);    //line121
   }
   else{ }
 }
 /* Check and give warnings if the vehicle is speeing on the 3rd scenario */
 else if(sector == 5 || sector == 6) {
   /* Detect if the velocity of the vehicle is just across 70 km/h */
   if(detectLastVelocity <= 70 && detectCurrentVelocity > 70)
        numCompare1++;
    /* Display the warning message on the screen */
    //DATAMETRIC::SetFeedbackMessageEvent("Speeding", "-5",0.5,0.5,0.5);
 //line131
 }
 else {
   /* Detect if the velocity of the vehicle is just across 40 km/h */
   if(detectLastVelocity < 40 && detectCurrentVelocity >= 40)
    if(detectLastVelocity <= 40 && detectCurrentVelocity > 40) {
         numCompare1++;
        /* Display the warning message on the screen */
         DATAMETRIC::SetFeedbackMessageEvent("Speeding", "-5",0.5,0.5,0.5);
    }
    else if(detectLastVelocity <= 45 && detectCurrentVelocity > 45) {                  //line141
         numCompare1++;
        /* Display the warning message on the screen */
            DATAMETRIC::SetFeedbackMessageEvent("Speeding", "-10",0.5,0.5,0.5);
    }
 /* Detect if the velocity of the vehicle is just across 55 km/h */
    else if(detectLastVelocity <= 55 && detectCurrentVelocity > 55) {
         numCompare1++;
        /* Display the warning message on the screen */
         DATAMETRIC::SetFeedbackMessageEvent("Speeding", "-15",0.5,0.5,0.5);
```

```
            }                                                                    //line151
          else { }
        }
        /* Check and give warnings if the vehicle is speeing on the 4th scenario */
        if(sector != last_sector) {
          if(sector == 5 || sector == 6) {
           /* Detect if the velocity of the vehicle is just across 70 km/h */
            if(detectCurrentVelocity > 70) numCompare2++;
          }
           /* Detect if the velocity of the vehicle is just across 40 km/h */
          else {                                                                 //line161
            if(detectCurrentVelocity > 40) numCompare2++;
          }
        }
        /* Compare the number ot times the vehicle is driving over the speed limit by two different
        Strategies, get the bigger number between the two */
        numOfSpeeding = (numCompare1 > numCompare2) ? numCompare1 : numCompare2;
        //std::cout << "The number of Speeding is: " << numOfSpeeding << std::endl;
        output_data["numOfSpeeding"] = numOfSpeeding;
        /* Get the number of wheels that are running on the road from the database */
           currentDrivingOffRoad = DATAMETRIC::GetLastInColumn("offRoad");              //line171
        lastDrivingOffRoad = DATAMETRIC::GetNextLastInColumn("offRoad");
        /* Check the status when the vehicle is driving off the road */
        if(currentDrivingOffRoad != 4 && lastDrivingOffRoad == 4) {
          numDrivingOffRoad++;
         /* Display warning message on the screen and deduct points from the total score at the same time */
          DATAMETRIC::SetFeedbackMessageEvent("Driving Off Road", "-5",0.5,0.5,0.5);
        }
        /* Save the number of times the vehicle has driven off the road to the database
        output_data["numOfDrivingOffRoad"] = numDrivingOffRoad;                   //line181

           /* Add num of times the vehicle is driving across the double yellow line */
        currentDrivingAcrossLine = DATAMETRIC::GetLastInColumn("LeftOrRight");
        lastDrivingAcrossLine = DATAMETRIC::GetNextLastInColumn("LeftOrRight");
    if(currentDrivingAcrossLine == 1 && lastDrivingAcrossLine == -1) {
          numDrivingAcrossLine++;
         /* Display warning messages on the screen */
          DATAMETRIC::SetFeedbackMessageEvent("Driving Across Line", "-5",0.5,0.5,0.5);
        }
        /* Save the number of times the vehicle is driving across the double yellow line to the database */
        output_data["numOfDrivingAcrossLine"] = numDrivingAcrossLine;                  //line191
        /* Calculate the total score by substracting the three factors */
        totalscore1 = s1 + s2 + s3 + s4 - 5 * (numDrivingOffRoad + numDrivingAcrossLine +
numOfSpeeding);
        stringstream ss;
        //ss  << "The final score is: " << s1 << "+" << s2 << "+" << s3 << "+" << s4 << "-" << "Penalties "
<< "=" << totalscore1 <<endl;

/* Give driver's rating based on their driving score */
    string DR;
        /* The driver's rating is according to the table on my 1st paper */
        if(totalscore1 >=90 && totalscore1 <= 100)                                     //line201
                DR = "excellent driver.";
        else if(totalscore1 >= 80 && totalscore1 < 90)
                DR = "good driver.";
        else if(totalscore1 >= 70 && totalscore1 < 80)
                DR = "fair driver.";
        else if(totalscore1 >= 60 && totalscore1 < 70)
                DR = "poor driver.";
        else
                DR = "dangerous driver.";
    ss << "You are a " << DR;                                       //line211
        /* Detect if the vehicle has finished one lap */
        if((sector != last_sector) && (last_sector == 9) )
        {
                /* Display final driving score on the screen */
                    DATAMETRIC::SetFeedbackMessageEvent("Scoring System",ss.str(),2.0,1.0,2.0);
        }
        // Output variables go here
        /* Save the total driving score to the database */
        output_data["TotalScore"] = totalscore1;

//1st Scenario                                                      //line221
        /* Get the score from the database */
        DATALOG::log_column_T const* score1_1_col = DATAMETRIC::GetColumn("score1_1");
        DATALOG::log_column_T const* score1_2_col = DATAMETRIC::GetColumn("score1_2");
        DATALOG::log_data_T s1_1 = 0.0;
        DATALOG::log_data_T s1_2 = 0.0;
        s1_1 = score1_1_col->back();
        s1_2 = score1_2_col->back();
/* Get params for 1st scenario */
   DATALOG::log_column_T const* param1_1_col = DATAMETRIC::GetColumn("param1_1");
   DATALOG::log_column_T const* param1_2_col = DATAMETRIC::GetColumn("param1_2");//line231
   DATALOG::log_column_T const* param1_3_col = DATAMETRIC::GetColumn("param1_3");
   DATALOG::log_column_T const* param1_4_col = DATAMETRIC::GetColumn("param1_4");
   /* Initialize the value to zero */
   DATALOG::log_data_T p1_1 = 0.0;
   DATALOG::log_data_T p1_2 = 0.0;
   DATALOG::log_data_T p1_3 = 0.0;
   DATALOG::log_data_T p1_4 = 0.0;
  /* Get value from column */
  p1_1 = param1_1_col->back();
   p1_2 = param1_2_col->back();                                     //line241
   p1_3 = param1_3_col->back();
   p1_4 = param1_4_col->back();


/* Get parameters for the 2nd Scenario */
        DATALOG::log_column_T const* score2_1_col = DATAMETRIC::GetColumn("score2_1");
        DATALOG::log_column_T const* score2_2_col = DATAMETRIC::GetColumn("score2_2");
        DATALOG::log_column_T const* score2_3_col = DATAMETRIC::GetColumn("score2_3");
        DATALOG::log_column_T const* score2_4_col = DATAMETRIC::GetColumn("score2_4");
        /* Set all value to zero */
          DATALOG::log_data_T s2_1 = 0.0;
        DATALOG::log_data_T s2_2 = 0.0;                             //line251
        DATALOG::log_data_T s2_3 = 0.0;
        DATALOG::log_data_T s2_4 = 0.0;
        /* Set all parameters with its number from the column */
        s2_1 = score2_1_col->back();
        s2_2 = score2_2_col->back();
        s2_3 = score2_3_col->back();
        s2_4 = score2_4_col->back();
   /* Get columns of the params for 2nd scenario from the database */
   DATALOG::log_column_T const* param2_1_col = DATAMETRIC::GetColumn("param2_1");
   DATALOG::log_column_T const* param2_2_col = DATAMETRIC::GetColumn("param2_2");//line261
   DATALOG::log_column_T const* param2_3_col = DATAMETRIC::GetColumn("param2_3");
   DATALOG::log_column_T const* param2_4_col = DATAMETRIC::GetColumn("param2_4");
   DATALOG::log_column_T const* param2_5_col = DATAMETRIC::GetColumn("param2_5");
   DATALOG::log_column_T const* param2_6_col = DATAMETRIC::GetColumn("param2_6");
   DATALOG::log_column_T const* param2_7_col = DATAMETRIC::GetColumn("param2_7");
   DATALOG::log_data_T p2_1 = 0.0;
   DATALOG::log_data_T p2_2 = 0.0;
   DATALOG::log_data_T p2_3 = 0.0;
   DATALOG::log_data_T p2_4 = 0.0;
   DATALOG::log_data_T p2_5 = 0.0;                                    //line271
   DATALOG::log_data_T p2_6 = 0.0;
   DATALOG::log_data_T p2_7 = 0.0;
   p2_1 = param2_1_col->back();
   p2_2 = param2_2_col->back();
   p2_3 = param2_3_col->back();
   p2_4 = param2_4_col->back();
   p2_5 = param2_5_col->back();
   p2_6 = param2_6_col->back();
   p2_7 = param2_7_col->back();


/* Get columns of the params for 2nd scenario from the database */              //line281
        DATALOG::log_column_T const* score3_1_col = DATAMETRIC::GetColumn("score3_1");
        DATALOG::log_column_T const* score3_2_col = DATAMETRIC::GetColumn("score3_2");
        DATALOG::log_column_T const* score3_3_col = DATAMETRIC::GetColumn("score3_3");
        DATALOG::log_column_T const* score3_4_col = DATAMETRIC::GetColumn("score3_4");
        DATALOG::log_data_T s3_1 = 0.0;
        DATALOG::log_data_T s3_2 = 0.0;
        DATALOG::log_data_T s3_3 = 0.0;
        DATALOG::log_data_T s3_4 = 0.0;
        s3_1 = score3_1_col->back();
        s3_2 = score3_2_col->back();                                  //line291
        s3_3 = score3_3_col->back();
        s3_4 = score3_4_col->back();
/* Get params for 3rd scenario */
   DATALOG::log_column_T const* param3_1_col = DATAMETRIC::GetColumn("param3_1");
   DATALOG::log_column_T const* param3_2_col = DATAMETRIC::GetColumn("param3_2");
   DATALOG::log_column_T const* param3_3_col = DATAMETRIC::GetColumn("param3_3");
   DATALOG::log_column_T const* param3_4_col = DATAMETRIC::GetColumn("param3_4");
   DATALOG::log_data_T p3_1 = 0.0;
   DATALOG::log_data_T p3_2 = 0.0;
   DATALOG::log_data_T p3_3 = 0.0;                                    //line301
   DATALOG::log_data_T p3_4 = 0.0;
   p3_1 = param3_1_col->back();
   p3_2 = param3_2_col->back();
   p3_3 = param3_3_col->back();
   p3_4 = param3_4_col->back();
/* Get params for 3rd scenario */
           DATALOG::log_column_T const* score4_1_col = DATAMETRIC::GetColumn("score4_1");
           DATALOG::log_column_T const* score4_2_col = DATAMETRIC::GetColumn("score4_2");
           DATALOG::log_column_T const* score4_3_col = DATAMETRIC::GetColumn("score4_3");
           DATALOG::log_column_T const* score4_4_col = DATAMETRIC::GetColumn("score4_4");
           DATALOG::log_data_T s4_1 = 0.0;                                          //line311
           DATALOG::log_data_T s4_2 = 0.0;
           DATALOG::log_data_T s4_3 = 0.0;
           DATALOG::log_data_T s4_4 = 0.0;
           s4_1 = score4_1_col->back();
           s4_2 = score4_2_col->back();
           s4_3 = score4_3_col->back();
           s4_4 = score4_4_col->back();
  /* Get Params for 4th scenario */
   DATALOG::log_column_T const* param4_1_col = DATAMETRIC::GetColumn("param4_1");
   DATALOG::log_column_T const* param4_2_col = DATAMETRIC::GetColumn("param4_2");/line321
   DATALOG::log_column_T const* param4_3_col = DATAMETRIC::GetColumn("param4_3");
   DATALOG::log_column_T const* param4_4_col = DATAMETRIC::GetColumn("param4_4");
   DATALOG::log_column_T const* param4_5_col = DATAMETRIC::GetColumn("param4_5");
   DATALOG::log_column_T const* param4_6_col = DATAMETRIC::GetColumn("param4_6");
   DATALOG::log_column_T const* param4_7_col = DATAMETRIC::GetColumn("param4_7");
   DATALOG::log_data_T p4_1 = 0.0;
   DATALOG::log_data_T p4_2 = 0.0;
   DATALOG::log_data_T p4_3 = 0.0;
   DATALOG::log_data_T p4_4 = 0.0;
   DATALOG::log_data_T p4_5 = 0.0;                                                  //line331
   DATALOG::log_data_T p4_6 = 0.0;
   DATALOG::log_data_T p4_7 = 0.0;
   p4_1 = param4_1_col->back();
   p4_2 = param4_2_col->back();
   p4_3 = param4_3_col->back();
   p4_4 = param4_4_col->back();
   p4_5 = param4_5_col->back();
   p4_6 = param4_6_col->back();
   p4_7 = param4_7_col->back();

/* Get some general values and save them in the matlab folder */                        //line341
if((sector != last_sector) && (last_sector == 9) )
{
 /* Get the percent of the speed over 60mph among all the speed (use the datacolVelocity above for
convenience)  and create a chart table in Matlab */
           /* Get the velocity from the database */
             DATALOG::log_column_T const* datacolVelocity2 = DATAMETRIC::GetColumn("Velocity");
           /* Count the total number of datas of the velocity in the column */
             double totalNumber = datacolVelocity2->size();
//         cout << "This is a test and the total number is:" << totalNumber << endl;
           double count60MPH = 0.0;                                              //line351
           double count30MPH = 0.0;
           double countOK = 0.0;
           /* Count the number of times the vehicle is driving over 60 km/h and between 30 and 60 km/h */
```

```
    for(DATALOG::log_column_T::const_iterator iterVelocity = datacolVelocity2-
>begin();iterVelocity != datacolVelocity2->end(); ++iterVelocity)
    {
        if((*iterVelocity)*2.2369 > 60)
          count60MPH++;
        else if( ((*iterVelocity)*2.2369 < 60) && ((*iterVelocity)*2.2369 > 30) )
          count30MPH++;                                               //line361
        else
          countOK++;
    }
    /* Calculate the perentage for veclocity between 60 and 30 among the whole lap driving */
    double rate60MPH = count60MPH/totalNumber;
    double rate30MPH = count30MPH/totalNumber;
    double rateOK = countOK/totalNumber;
    stringstream rateStr;
    rateStr << rate60MPH << "," << rate30MPH << "," << rateOK;
    /* Save the percentage to the matlab dat file */                          //line371
      ofstream outFileRate("/home/yaoqimin/matlab/bin/velocityRate.dat");
    if(outFileRate.is_open())
    {
        outFileRate << rateStr.str();
    }
    outFileRate.close();
    /* Get the perentage of the number of times the vehicle is driving off the road */
    DATALOG::log_column_T const* datacoloffRoad = DATAMETRIC::GetColumn("offRoad");
    /* Count the total number of the data, drivingOff road */
      double allNumber = datacoloffRoad->size();                                     //line381
    double onRoad = 0.0;
    double offRoad = 0.0;
    double twoWheelsOff = 0.0;
    /* Count the number of times the vehicle is on the road, two wheels off, and driving off the road */
    for(DATALOG::log_column_T::const_iterator it = datacoloffRoad->begin(); it != datacoloffRoad-
>end(); ++it)
    {
        if(*it == 4)
          onRoad++;
        else if(*it == 0)                                                     //line391
          offRoad++;
        else
          twoWheelsOff++;
    }
    /* Calculate the percentage of the above three parameters */
    double onRoadRate = onRoad/allNumber;
    double offRoadRate = offRoad/allNumber;
    double twoWheelsOffRate = twoWheelsOff/allNumber;
    stringstream roadRateStr;
    roadRateStr << onRoadRate << "," << offRoadRate << "," << twoWheelsOffRate;      //line401
    /* Save the results to a Matlab data type file */
      ofstream outFileRoadRate("/home/yaoqimin/matlab/bin/offRoadRate.dat");
    if(outFileRoadRate.is_open())
    {
        outFileRoadRate << roadRateStr.str();
    }
```

```
        /* Close the file */
        outFileRoadRate.close();

/* Calculate the number of times the vehicle is driving off the road */
        double countOffRoadNumber = 0;                                          //line411
        for(DATALOG::log_column_T::const_iterator it2 = datacoloffRoad->begin(); it2 !=
(datacoloffRoad->end() - 1); ++it2) {
                if( (*it2 != 4) && (*(it2+1) == 4) ) {
                  countOffRoadNumber++;
                }
        }
/* Calculate the number of times the vehicle is driving across the double yellow line */
        double countCrossLineNumber = 0;
        DATALOG::log_column_T const* datacolCrossLine =
DATAMETRIC::GetColumn("LeftOrRight");                                           //line421
        for(DATALOG::log_column_T::const_iterator it3 = datacolCrossLine->begin(); it3 !=
(datacolCrossLine->end() - 1); ++it3) {
                if( (*it3 == 1) && (*(it3+1) == -1) ) {
                        countCrossLineNumber++;
                }
        }

         /* Save data to the database */
        vector<string> v;
        ifstream inFile;
        inFile.open("dataTest.csv");                                    //line431

        if(!inFile)
        {
                cout << "Could not open file" << endl;
                inFile.open("initialDataTest.csv");
        }
        /* Read file and get each line the excel file */
        while(!inFile.eof())
        {
                string oneLine;
                /* Get each line */                                     //line441
                getline(inFile,oneLine);
                /* Add each line to the vector */
                v.push_back(oneLine);
        }
        /* Close file */
        inFile.close();
        /* Manually change the size of the file */
        v.resize(100);

        /* Add numb of time the vehicle is driving offRoad and crossLine here */
        //if((int(lap_number) + 1) % 2 == 0) {                                  //line451
        stringstream nfr; // number of off road
        nfr << countOffRoadNumber * (-20) << ";";
        /* Add the message into the container vector */
        v[26] += nfr.str();
        stringstream ncl; // number of cross line
        ncl << countCrossLineNumber * (-20) << ";";
        /* Add the message into the container vector */
          v[27] += ncl.str();

          /* Add the number of times the vehicle is driving over the speed limit, and save
             The result to the container */                                        //line461
        stringstream nsp; // number of speeding
        nsp << numOfSpeeding * (-20) << ";";
        v[28] += nsp.str();

        /* Normalize the score */
        stringstream ssRaw;
        ssRaw << totalscore1 * 4 << "/400;";
        v[25] += ssRaw.str();
        /* Calculate the final score */
        totalscore1 = totalscore1  - countOffRoadNumber * 5 - countCrossLineNumber * 5;
          /* Add the number of times the vehicle is speeding */                          //line471
   //if(numCompare1 > numCompare2) numOfSpeeding = numCompare1;
        //else   numOfSpeeding = numCompare2;

        totalscore1 = totalscore1 * 4  - countOffRoadNumber * 20 - countCrossLineNumber * 20 -
numOfSpeeding * 20;


        /* Print out the result */
          std::cout << "The number of compare1 is: " << numCompare1 << std::endl;
        std::cout << "The number of compare2 is: " << numCompare2 << std::endl;
        std::cout << "The number of speeding is: " << numOfSpeeding << std::endl;
        std::cout << "The final score combined with speeding is: " << totalscore1 << std::endl;
        stringstream ss;                                                           //line481
        ss << totalscore1 << "/400;";

        v[29] += ss.str();

        stringstream ss2;
        ss2 << totalscore1/4 << "/100;";
        v[30] += ss2.str();
        /* Give the driver's rating based on the follow equations */
        totalscore1 /= 4;
        if(totalscore1 >=90 && totalscore1 <= 100)
                v[31] += "Excellent driver;";
        else if(totalscore1 >= 80 && totalscore1 < 90)                              //line491
                v[31] += "Good driver;";
        else if(totalscore1 >= 70 && totalscore1 < 80)
                v[31] += "Fair driver;";
        else if(totalscore1 >= 60 && totalscore1 < 70)
                v[31] += "Poor driver;";
        else
                v[31] += "Dangerous driver;";

  /* Add parameters for 1st scenario */
   stringstream ps1_1; // p stream
```

```
        ps1_1 << p1_1 << ";";
        v[65] += ps1_1.str();                                          //line501

        stringstream ps1_2;
        ps1_2 << p1_2 << ";";
        v[66] += ps1_2.str();

        stringstream ps1_3;
        ps1_3 << p1_3 << ";";
        v[67] += ps1_3.str();

        stringstream ps1_4;
        ps1_4 << p1_4 << ";";
        v[68] += ps1_4.str();
 /* Add parameters for 2nd scenario */                                     //line511
 stringstream ps2_1;
        ps2_1 << p2_1 << ";";
        v[70] += ps2_1.str();

 stringstream ps2_2;
        ps2_2 << p2_2 << ";";
        v[71] += ps2_2.str();

 stringstream ps2_3;
        ps2_3 << p2_3 << ";";
        v[72] += ps2_3.str();

 stringstream ps2_4;                                                       //line521
        ps2_4 << p2_4 << ";";
        v[73] += ps2_4.str();

 stringstream ps2_5;
        ps2_5 << p2_5 << ";";
        v[74] += ps2_5.str();

 stringstream ps2_6;
        ps2_6 << p2_6 << ";";
        v[75] += ps2_6.str();

 stringstream ps2_7;
        ps2_7 << p2_7 << ";";                                              //line531
        v[76] += ps2_7.str();

/* Add parameters for 3rd scenario */
 stringstream ps3_1; // p stream
        ps3_1 << p3_1 << ";";
        v[78] += ps3_1.str();

        stringstream ps3_2;
        ps3_2 << p3_2 << ";";
        v[79] += ps3_2.str();

        stringstream ps3_3;
        ps3_3 << p3_3 << ";";                                           //line541
        v[80] += ps3_3.str();

        stringstream ps3_4;
        ps3_4 << p3_4 << ";";
        v[81] += ps3_4.str();
    /* Add parameters for 4th scenario */
    stringstream ps4_1;
        ps4_1 << p4_1 << ";";
        v[83] += ps4_1.str();

    stringstream ps4_2;
        ps4_2 << p4_2 << ";";                                           //line551
        v[84] += ps4_2.str();

    stringstream ps4_3;
        ps4_3 << p4_3 << ";";
        v[85] += ps4_3.str();

    stringstream ps4_4;
        ps4_4 << p4_4 << ";";
        v[86] += ps4_4.str();

    stringstream ps4_5;
        ps4_5 << p4_5 << ";";
        v[87] += ps4_5.str();                                       //line561

    stringstream ps4_6;
        ps4_6 << p4_6 << ";";
        v[88] += ps4_6.str();

    stringstream ps4_7;
        ps4_7 << p4_7 << ";";
        v[89] += ps4_7.str();
        /* Add General Information --- Velocity */
        DATALOG::log_column_T const* datacolVelocity = DATAMETRIC::GetColumn("Velocity");
        double maxVelocity = 0.0;
        double averageVelocity = 0.0;                                   //line571
        double sum = 0.0;
        int count = 0;
        //int numberSpeeding = 0; // Speed over 50 mph
        for(DATALOG::log_column_T::const_iterator iterVelocity = datacolVelocity->begin();iterVelocity
!= datacolVelocity->end(); ++iterVelocity)
        {
                if(maxVelocity < *iterVelocity)
                  maxVelocity = *iterVelocity;
                DATALOG::log_data_T value = *iterVelocity;
                sum += value;                                       //line581
                count++;
        //      if(value > 50) {
        //         numberSpeeding++;
        //      }
        }
```

```
	maxVelocity *=2.2369;
	averageVelocity = (sum / count) * 2.2369;
	maxVelocity = int((maxVelocity + 0.05) * 10)/10.0;
	averageVelocity = int((averageVelocity + 0.05) * 10)/10.0;

	stringstream tt;                                              //line591
	tt << maxVelocity << ";";

	v[91] += tt.str();

	stringstream vv;
	vv << averageVelocity << ";";
	v[92] += vv.str();

	//stringstream ns; // number speeding
	//ns << numberSpeeding << ";";
	//v[88] += ns.str();
	/* Add General Information – Steering angle */
	DATALOG::log_column_T const* datacolSteeringAngle =                //line601
		DATAMETRIC::GetColumn("Steering");
	double sumSteeringAngle = 0.0;
	double averageSteeringAngle = 0.0;
	double maxSteeringAngle = 0.0;
	int countAngle = 0;
	for(DATALOG::log_column_T::const_iterator iterAngle = datacolSteeringAngle->begin();
iterAngle != datacolSteeringAngle->end(); ++iterAngle) {
		if(maxSteeringAngle < *iterAngle) {
			maxSteeringAngle = *iterAngle;
		}                                                     //line611
		DATALOG::log_data_T valueAngle = *iterAngle;
		sumSteeringAngle += valueAngle;
		countAngle++;
	}
	averageSteeringAngle = sumSteeringAngle / countAngle;
	averageSteeringAngle = int((averageSteeringAngle + 0.05) * 10) / 10.0;
	maxSteeringAngle = int((maxSteeringAngle + 0.05) * 10) / 10.0;

	stringstream s_max_angle;
	s_max_angle << maxSteeringAngle << ";";
	v[93] += s_max_angle.str();                                   //line621

	stringstream s_average_angle;
	s_average_angle << averageSteeringAngle << ";";
	v[94] += s_average_angle.str();

	stringstream s_sum_angle;
	s_sum_angle << sumSteeringAngle << ";";
	v[95] += s_sum_angle.str();
	/* Add General Information -- Steering Rate */
	DATALOG::log_column_T const* datacolSteeringRate =
DATAMETRIC::GetColumn("SteeringRate");
	double sumSteeringRate = 0.0;                                     //line631
	double averageSteeringRate = 0.0;
```

```
    double maxSteeringRate = 0.0;
    int countRate = 0;
    for(DATALOG::log_column_T::const_iterator iterRate = datacolSteeringRate->begin(); iterRate !=
datacolSteeringRate->end(); ++iterRate) {
        if(maxSteeringRate < *iterRate) {
            maxSteeringRate = *iterRate;
        }
        DATALOG::log_data_T valueRate = *iterRate;
        sumSteeringRate += valueRate;                                   //line641
        countRate++;
    }
    averageSteeringRate = sumSteeringRate / countRate;
    averageSteeringRate = int((averageSteeringRate + 0.05) * 10) / 10.0;
    maxSteeringRate = int((maxSteeringRate + 0.05) * 10) / 10.0;

      /* Write data to dataTest.csv */
    vector<string>::iterator it;

    ofstream outFile("dataTest.csv");
    if(outFile.is_open())
    {                                                                   //line651
        for(it = v.begin(); it != v.end(); it++)
          outFile << *it << endl;
    }
    outFile.close();

    /* Get x coordinate of car position and write it to Matlab */
    stringstream dd;
    DATALOG::log_column_T const* positionX = DATAMETRIC::GetColumn("CarPositionX");
    DATALOG::log_column_T::const_iterator iit;
    dd << *(positionX->begin());
    for(iit = positionX->begin(); iit != positionX->end(); ++iit)          //line661
    {
        if(*iit)
            dd << "," << *iit;
    }


    ofstream outFileX("/home/yaoqimin/matlab/bin/positionX.dat");
    if(outFileX.is_open())
    {
        outFileX << dd.str();
    }
    outFileX.close();                                                   //line671
    // Y position
    stringstream ee;
    DATALOG::log_column_T const* positionY = DATAMETRIC::GetColumn("CarPositionY");
    DATALOG::log_column_T::const_iterator iity;
    ee << *(positionY->begin());
    for(iity = positionY->begin(); iity != positionY->end(); ++iity)
    {
        if(*iity)
            ee << "," << *iity;
    }                                                                              //line681

    /* Get y coordinate of car position and write it to Matlab */
    ofstream outFileY("/home/yaoqimin/matlab/bin/positionY.dat");
    if(outFileY.is_open())
    {
        outFileY << ee.str();
    }
    outFileY.close();
    //Z position
    /* Get z coordinate of car position and write it to Matlab */
    stringstream cz;                                                                     //line691
    DATALOG::log_column_T const* positionZ = DATAMETRIC::GetColumn("CarPositionZ");
    DATALOG::log_column_T::const_iterator iitz;
    cz << *(positionZ->begin());
    for(iitz = positionZ->begin(); iitz != positionZ->end(); ++iitz)
    {
        if(*iitz)
            cz << "," << *iitz;
    }
    ofstream outFileZ("/home/yaoqimin/matlab/bin/positionZ.dat");
    if(outFileZ.is_open())                                                               //line701
    {
        outFileZ << cz.str();
    }
    outFileZ.close();
    cout << "3" << endl;

    /* Get velocity values and save them to Matlab */
    stringstream vvv;
    DATALOG::log_column_T const* velocity_column = DATAMETRIC::GetColumn("Velocity");
    DATALOG::log_column_T::const_iterator iitV;
    vvv << *(velocity_column->begin());                                                  //line711
    for(iitV = velocity_column->begin(); iitV != velocity_column->end(); ++iitV)
    {
        if(*iitV)
        {
            vvv << "," << (*iitV)*2.2369;
        }
    }


    ofstream outFileVelocity("/home/yaoqimin/matlab/bin/velocity.dat");
    if(outFileVelocity.is_open())
    {                                                                              //line721
        outFileVelocity << vvv.str();
    }
    outFileVelocity.close();
    cout << "0" << endl;

    /* Get time values and save them to the matlab folder */
    DATALOG::log_column_T const* datacolTime = DATAMETRIC::GetColumn("Time");
    DATALOG::log_column_T::const_iterator itTime;
```

```
stringstream timeStr;
timeStr << *(datacolTime->begin());
for(itTime = datacolTime->begin(); itTime != datacolTime->end();++itTime)          //line731
{
        if(*itTime)
          timeStr << "," << *itTime;
}

ofstream outFileTime("/home/yaoqimin/matlab/bin/time.dat");
if(outFileTime.is_open())
{
        outFileTime << timeStr.str();
}
outFileTime.close();                                                   //line741

/* Get Lateral Velocity and send them to Matlab */
DATALOG::log_column_T const* datacolLV = DATAMETRIC::GetColumn("LateralVelocity");
DATALOG::log_column_T::const_iterator itLV;
stringstream LVStr;
LVStr << *(datacolLV->begin());
for(itLV = datacolLV->begin(); itLV != datacolLV->end();++itLV)
{
        if(*itLV)
          LVStr << "," << *itLV;
        Else                                                           //line751
          LVStr << "," << 0;
}

ofstream outFileLV("/home/yaoqimin/matlab/bin/lateralVelocity.dat");
if(outFileLV.is_open())
{
        outFileLV << LVStr.str();
}
outFileLV.close();

/* Get Steering Angle and send them to the Matlab */                              //line761
DATALOG::log_column_T const* datacolSteering = DATAMETRIC::GetColumn("Steering");
DATALOG::log_column_T::const_iterator itSteering;
stringstream SteeringStr;
SteeringStr << *(datacolSteering->begin());
for(itSteering = datacolSteering->begin(); itSteering != datacolSteering->end();++itSteering)
{
        if(*itSteering)
          SteeringStr << "," << *itSteering;
        else
          SteeringStr << "," << 0;                                            //line771
}

ofstream outFileSteering("/home/yaoqimin/matlab/bin/steering.dat");
if(outFileSteering.is_open())
{
        outFileSteering << SteeringStr.str();
}
```

```
        outFileSteering.close();

        /* Get Yaw Angle and put them to Matlab */
        DATALOG::log_column_T const* datacolYawAngle =
DATAMETRIC::GetColumn("YawAngle");                                              //line781
        DATALOG::log_column_T::const_iterator itYawAngle;
        stringstream YawAngleStr;
        YawAngleStr << *(datacolYawAngle->begin());
        for(itYawAngle = datacolYawAngle->begin(); itYawAngle != datacolYawAngle->end();
++itYawAngle)
        {
                if(*itYawAngle)
                  YawAngleStr << "," << *itYawAngle;
                else
                  YawAngleStr << "," << 0;                                      //line791
        }

        ofstream outFileYawAngle("/home/yaoqimin/matlab/bin/YawAngle.dat");
        if(outFileYawAngle.is_open())
        {
                outFileYawAngle << YawAngleStr.str();
        }
        outFileYawAngle.close();

        /* Get Lateral Acceleration and send them to Matlab folder */
        DATALOG::log_column_T const* datacolLateralAcceleration =
DATAMETRIC::GetColumn("LateralAcceleration");                                   //line801
        DATALOG::log_column_T::const_iterator itLateralAcceleration;
        stringstream LateralAccelerationStr;
        LateralAccelerationStr << *(datacolLateralAcceleration->begin());
        for(itLateralAcceleration = datacolLateralAcceleration->begin(); itLateralAcceleration !=
datacolLateralAcceleration->end(); ++itLateralAcceleration)
        {
                if(*itLateralAcceleration)
                  LateralAccelerationStr << "," << *itLateralAcceleration;
                else
                  LateralAccelerationStr << "," << 0;                           //line811
        }
        ofstream outFileLateralAcceleration("/home/yaoqimin/matlab/bin/LateralAcceleration.dat");
        if(outFileLateralAcceleration.is_open())
        {
                outFileLateralAcceleration << LateralAccelerationStr.str();
        }
        outFileLateralAcceleration.close();
 }
}
```

REFERENCES

"Bob's Track Builder", www.bobstrackbuilder.net, accessed July 2012.

Chan, C., Pradhan, A., Pollatsek, A., Knodler, M., and Fisher, D., "Are Driving Simulators Effective Tools for Evaluating Novice Drivers' Hazard Anticipation, Speed Management, and Attention Maintenance Skills?", Transportation Research Part F, vol. 13, no. 5, pp. 343-353, September 2010.

Christian, L., Hanna, Laflamme, L., and Raymond, C., "Fatal Crash Involvement of Unlicensed Young Drivers: County Level Differences According to Material Deprivation and Urbanicity in the United States", Journal of Accident Analysis and Prevention, vol. 45, pp. 291-295, March 2012.

Craen, S., Twisk, D., Hagenzieker, M., Elffers, H., and Brookhuis, K., "Do Young Novice Drivers Overestimate Their Driving Skills More Than Experienced Drivers? Different Methods Lead to Different Conclusion", *Journal of Accident Analysis and Prevention,* vol. 43, no. 5, pp.1660-1665, September 2011.

Crundall, D., Andrews, B., Loon, E., and Chapman, P., "Commentary Training Improves Responsiveness to Hazards in a Driving Simulator", Accident Analysis and Prevention, vol. 42, no. 6, pp. 2117-2124, November 2010.

Deery, A., and Fildes, B., "Young Novice Driver Subtypes: Relationship to High-Risk Behavior, Traffic Accident Record, and Simulator Driving Performance", The Journal of the Human Factors and Ergonomics Society, vol. 41, no. 4, pp. 628-643, December 1999.

Deery, H., "Hazard and Risk Perception Among Young Novice Drivers", Journal of Safety Research, vol. 30, no. 4, pp. 225-236, December 1999.

Dutta, A., Fisher, D., and Noyce, D, "Use of a Driving Simulator to Evaluate and Optimize Factors Affecting Understandability of Variable Message Signs", Transportation Research Part F, vol. 7, no. 4-5, pp. 209-227, July-September 2004.

Dols, J., Pardo, J., Verwey, W., and Ward, D., "Trainer Project: Development of an Improved Learning Method for Training Novice Drivers with Simulators", proceeding of the Automotive and Transportation Technology Congress, SAE Techanical Paper 2001-01-3381, Barcelona, Spain, October 2001.

Ferguson, S., "Other High-Risk Factors for Young Drivers–How Graduated Licensing Does, Doesn't, or Could Address Them", Journal of Safety Research, vol. 34, no. 1, pp. 71-77, January 2003.

Green, P., “Driver Distraction/Overload Research and Engineering: Problems and Solutions”, proceeding of the SAE Convergence Conference, SAE Technical Paper 2010-10-19, Detroit, MI, October 2010.

Groot, S., Ricote, F., and Winter, J., “The effect of tire grip on learning driving skill and driving style: A driving simulator study”, Journal of Transportation Research, vol. 15, no., pp. 413-426, February 2012.

Kandhai, K.; Smith, M.; Kanneh, A., "Immersive driving simulation for driver education and analysis," Computer Games (CGAMES), 2011 16th International Conference on , vol., no., pp.288-292, 27-30 July 2011

Kemeny, A., and Panerai, F., “Evaluating Perception in Driving Simulation Experiments”, *Journal of Trends in Cognitive Sciences*, vol. 7, no. 1, pp. 31-37, January 2003.

Kim, M., Lee, S., and Yu, S., “Development of a Vehicle Simulator Based Testing Method for Telematics Software Development”, proceeding of the 2007 SAE World Congress and Exhibition, SAE Technical Paper 2007-01-0945, Detroit, MI, April 2007.

Lee, J., “Technology and Teen Drivers”, Journal of Safety Research, vol. 38, no. 2, pp. 203-213, March 2007.

Lonero, L., “Trends in Driver Education and Training”, American Journal of Preventive Medicine, vol. 35, no. 3, pp. 316-323, September 2008.

McCartt, T., Mayhew, R., Braitman, A., Ferguson, A., and Simpson, M., “Effects of Age and Experience on Young Driver Crashes: Review of Recent Literature”, *Journal of Traffic Injury Prevention*, vol. 10, no. 3, pp. 209-219, May 2009.

Muttart, J., Fisher, D., Knodler, M., and Pollatsek, A., “Evaluation of Driver Simulator Performance During Hands-Free Cell Phone Operation in a Work Zone”, Transportation Research Record: Journal of the Transportation Research Board, vol. 2018, pp. 9-14, January 2008.

National Highway Traffic Safety Administration (NHTSA), “Traffic Safety Fact Sheet 2010”, www.nhtsa.gov, September 2010.

National Highway Traffic Safety Administration (NHTSA), “Graduated Driver Licensing”, www.nhtsa.gov, September 2004.

Norfleet, D., Wagner, J., Alexander, K., and Pidgeon, P., "Automotive Driving Simulators: Re- search, Education, and Entertainment", proceeding of the 2009 SAE World Congress and Exhibition, SAE Technical Paper 2009-04-20, Detroit, MI, April 2009.

Norfleet, D., Wagner, J., Jensen, M., Alexander, K., and Pidgeon, P., "Automotive Simulator Based Novice Driver Training with Assessment", proceeding of the 2011 SAE World Congress and Exhibition, SAE Technical Paper 2011-04-12, Detroit, MI, April 2011.

Park, S., and Lim, H., "Characteristics of Elderly Driver's Driving Behavior and Cognition under Unexpected Event Using Driving Simulator", proceeding of the SAE World Congress, SAE Technical Paper 2011-04-12, Detroit, MI, April 2011.

Piegsa, A., Rumbolz, P., Schmidt, A., Liedecke, C., and Reuss, H., "VALIDATE-Basis for New Sophisticated Research Platform for Virtual Development of Vehicle Systems", proceeding of the 2011 SAE World Congress and Exhibition, SAE Technical Paper 2011-01-1012, Detroit, MI, April 2011.

Ruspa, C., Quattrocolo, S., and Bertolino, D., "Virtual Tool for the Evaluation of the Visibility during Critical Driving Tasks", proceeding of the 2007 SAE Digital Human Modeling for Design and Engineering Conference and Exhibition, SAE Technical Paper 2007-01-2499, Seattle, WA, June 2007.

Roosendaal, T., "Blender", www.blender.org, accessed July 2012.

Schultheis, M., Simone, L., Roseman, E., Nead, R., Rebimbas, J., and Mourant, R., "Stopping Behavior in a VR Driving Simulator: A New Clinical Measure for the Assessment of Driving", 28th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, New York, NY, pp. 4921-2924, August 2006.

Scialfa, C., Deschenes, M., Ference, J., Boone, J., Horswill, M., and Wetton, M., "A Hazard Perception Test for Novice Drivers", *Journal of Accident Analysis and Prevention*, vol. 45, no. 1, pp. 204-208, January 2011.

Vidotto, G., Bastianelli, A., Spoto, A., and Sergeys, F., "Enhancing Hazard Avoidance in Teen-Novice Riders", Accident Analysis and Prevention, vol.45, no. 1, pp. 247-252, January 2011.

"Vamos", www.vamos.sourceforge.net, accessed July 2012.

Venjon, J., "VDrift", www.vdrift.net, accessed July 2012.

Wahlberg, A., “Re-education of Young Driving Offenders, Effects on Self-Reports of Driver Behavior”, *Journal of Safety Research*, vol. 41, no. 4, pp. 331-338, August 2010.

Yao, Q., Wagner, J., Alexander, K., and Pidgeon, P., “A Virtual Driving Education Simulation System - Hardware, Software and Assessment”, submitted to the SAE World Congress, Detroit, MI, April 2013.